Thesis to get the bachelor's degree

Parallel implementation of the model linear problem solution using FETI-DP

Paralelní implementace řešení modelové lineární úlohy metodou FETI-DP

Jiří Tomčala

May 7-th 2013

VŠB - Technical University Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Applied Mathematics

"The wise know their limitations; the foolish do not."
Benjamin Hoff, The Tao of Pooh

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání bakalářské práce

Student: **Jiří Tomčala**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 1103R031 Výpočetní matematika

Téma: Paralelní implementace řešení modelové lineární úlohy metodou FETI-DP
Parallel implementation of the model linear problem solution using FETI-DP

Zásady pro vypracování:

Efektivní řešení soustav lineárních rovnic vzniknuvších konečněprvkovou diskretizací je základem řešení rozsáhlých inženýrských úloh většinou popsaných parciálními diferenciálními rovnicemi. Metody rozložení roblasti (Domain decomposition methods) FETI typu patří v dnešní době k těm nejefektivnějším. Jednou z nich je metoda FETI-DP (Dual-Primal) rozkládající oblast na nepřekrývajícíse podoblasti, jejichž spojitost na rozhraních je vynucována tzv. Lagrangeovými multiplikátory na duální úrovni a globálním vektorem tzv. cornerů, tj. rožních uzlů, na úrovni primární.
Cílem této diplomové práce je paralelně implemetovat tuto metodu prostřednictvím softwaru pro vědecké rozsáhlé výpočty PETSc z Argonne National Laboratory. Iterační metodou pro řešení duální úlohy by byla metoda sdružených gradientů (CG) s předpodmíněním (Dirichlet, lumped).
Systematicky zpracované experimenty by měly prokázat jak paralelní, tak i numerickou škálovatelnost tohoto algoritmu.
1. Seznámit se s FETI-DP.
2. Seznámit se s PETSc.
3. Paralelně implementovat algoritmus s využitím PETSc.
4. Systematicky realizovanými experimenty prokázat škálovatelnost algoritmu.
An efficient solution of linear systems of equations arising from finite element discretization is the basic tool for the solution of large scale engineering problems described by partial differential equations. The domain decomposition methods (DDM) of FETI type belong to the most efficient in this time. One of them is FETI-DP (Dual-Primal) decomposing the domain into non-overlapping subdomains and the continuity across interfaces is ensured by Lagrange multipliers as dual variables and by so called corners on primal level.
The aim of this thesis is parallel implementation of this method using software for scientific computing PETSc developed in Argonne National Laboratory. The iterative method for the dual problem solution is method of conjugate gradients (CG) with preconditioning (Dirichlet, lumped). The systematically collected numerical experiments should prove high parallel and numerical scalability of this algorithm.
1. to study FETI-DP
2. to get know PETSc
3. to implement FETI-DP in parallel using PETSc
4. to prove the algorithm's scalability by means of numerical experiments

Seznam doporučené odborné literatury:

[1] Z. Dostál, D. Horák a D. Stefanica: A scalable FETI-DP algorithm for a coercive variational inequalities, IMACS J. Appl. Numer. Math., 2005.
[2] Z. Dostál, D. Horák a D. Stefanica: An Overview of Scalable FETI-DP Algorithms for Variational

Inequalities, Lecture Notes in Comput. Science and Engineering 55, Proceedings from the 16th conference on DDM, New York, Springer, 2006, 223-230.
[3] Z. Dostál, D. Horák a D. Stefanica: A Scalable FETI--DP Algorithm for Semi-coercive Variational Inequality, Computer Methods in Applied Mechanics and Engineering, Vol. 196, 8, ISSN 0045-7825, 2007, 1369-1379.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Horák, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7.května 2013

.....

Jiří Tomčala

Contents

Abstract

This thesis deals with the parallel solution of partial differential equation by one of the most successful classes of methods called domain decomposition method.
It leads from Poisson equation through its weak formulation, Galerkin method and finite element method to domain decomposition method FETI-DP. It shows its parallel implementation as well as numerical experiments with parallel and numerical scalability and corner choosing strategy. This work also contains few simple examples and several benchmarks that illustrate the behaviour of the method, mainly its parallel and numerical scalabilities.

Tato práce se zabývá paralelním řešením parciální diferenciální rovnice jednou z nejúspěšnějších metod rozložení oblastí.
Vede čtenáře od Poissonovy rovnice, přes její slabou formulaci, Galerkinovu metodu a metodu konečných prvků až k metodě rozložení oblastí FETI-DP. Ukazuje její paralelní implementaci stejně jako numerické experimenty s paralelní a numerickou škálovatelností a strategiemi výběru rohových uzlů. Tato práce obsahuje také pár jednoduchých příkladů a několik benchmarků, které názorně ukazují vlastnosti metody, hlavně její paralelní a numerickou škálovatelnost.

1. Introduction

1.1. Poisson equation

When we are solving some simple example of body deformations, we can use Poisson equation. It is the simplest and the most famous elliptic partial differential equation. It looks like this :

$$-\triangle u = f \quad \text{in } \Omega, \tag{1.1}$$

where u is unknown displacement, f is load function or we can say impacting force and Ω is the considered domain.
If we add boundary condition of the Dirichlet type :

$$u = 0 \quad \text{on } \partial\Omega, \tag{1.2}$$

where $\partial\Omega$ is boundary of Ω, we get so called Dirichlet problem for the Poisson equation. To show some simple example of the analytical solution I wanted to avoid one-dimensional string (it's in every thesis) and show analytical solution of a two-dimensional membrane. Unfortunately I found out that it would take over couple of pages here, so in the end, it's old good string here :

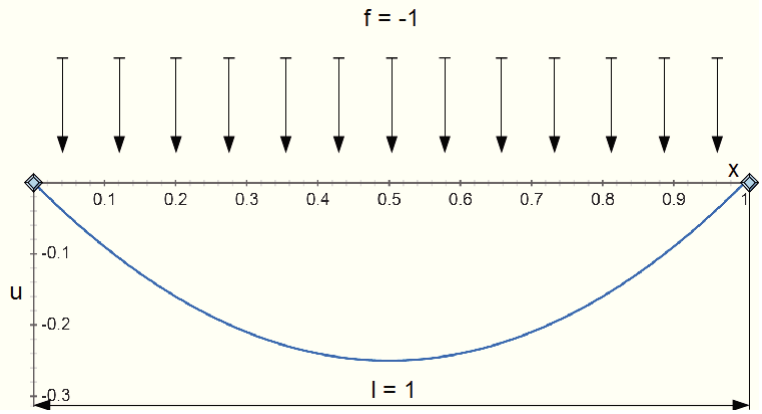


Figure 1.1.: One-dimensional example - the loaded string.

In this one-dimensional case we can write

$$-\triangle u = -\frac{\partial^2 u}{\partial x^2} = f. \tag{1.3}$$

Now, let's assume that f is constant and does not depend on x. Then we can write

$$\frac{\partial u}{\partial x} = \int -f dx = -fx + C_1, \tag{1.4}$$

$$u = \int \left(-fx + C_1\right) dx = -\frac{f}{2}x^2 + C_1 x + C_2. \tag{1.5}$$

Applying Dirichlet boundary conditions we get

$$u\left(0\right) = 0 \quad \Rightarrow \quad -\frac{f}{2}0^2 + C_1 0 + C_2 = 0 \quad \Rightarrow \quad C_2 = 0, \tag{1.6}$$

$$u\left(l\right) = 0 \quad \Rightarrow \quad -\frac{f}{2}l^2 + C_1 l = 0 \quad \Rightarrow \quad C_2 = \frac{1}{2}fl, \tag{1.7}$$

$$u = \frac{1}{2}f\left(-x^2 + lx\right). \tag{1.8}$$

[Str08c, Str08d]

1.2. Weak formulation

In real life we need to solve deformations of quite complicatedly shaped objects. Poisson equation is then really hard to solve, even in some cases impossible. Analytical solution of Poisson equation we call *strong solution*. It's the most exact we can get. But if we can not obtain it, we are looking for so called *weak solution*. What is it? Let's take a look. Weak solution is a solution of the *weak formulation* of a differential equation, in our case Poisson equation. Weak formulation is rewritten differential equation in such a way that no derivatives of the solution of the equation show up. How is that possible? Let's try to rewrite our Poisson differential equation to its weak formulation.

Here is n-dimensional version of Poisson equation

$$-\triangle u = \sum_{i=0}^{n-1} -\frac{\partial^2 u}{\partial x_i^2} = f. \tag{1.9}$$

Now, we multiply both sides by some test function v (it will be later explained why)

$$\sum_{i=0}^{n-1} -\frac{\partial^2 u}{\partial x_i^2}\, v = fv \tag{1.10}$$

and integrate whole equation in domain Ω

$$\int_\Omega \left(\sum_{i=0}^{n-1} -\frac{\partial^2 u}{\partial x_i^2}\, v \right) \mathrm{d}\Omega = \int_\Omega fv \,\mathrm{d}\Omega. \tag{1.11}$$

Let us keep right side as it is and take care of left side. First, we can change order of integration and sum. It is possible, because we expect that the sum convergates equally

$$\sum_{i=0}^{n-1} \int_\Omega -\frac{\partial^2 u}{\partial x_i^2}\, v \,\mathrm{d}\Omega = \int_\Omega fv \,\mathrm{d}\Omega. \tag{1.12}$$

Now, we can integrate left side "per partes"

$$\sum_{i=0}^{n-1} \int_\Omega \left(\frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_i} - \frac{\partial}{\partial x_i}\left(\frac{\partial u}{\partial x_i}\, v \right) \right) \mathrm{d}\Omega = \int_\Omega fv \,\mathrm{d}\Omega. \tag{1.13}$$

And if we apply "magic" of Green theorem, we get

$$\sum_{i=0}^{n-1} \left(\int_\Omega \frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_i} \,\mathrm{d}\Omega - \oint_{\partial\Omega} \frac{\partial u}{\partial \nu}\, v \,\mathrm{d}s \right) = \int_\Omega fv \,\mathrm{d}\Omega, \tag{1.14}$$

where ν is direction allways perpendicular to $\partial\Omega$, the boundary of Ω.
Now it's time to reveal why we multiplicated equation by test function v before. If we choose function v so, that its value in boundary of Ω is zero, then

$$\oint_{\partial\Omega} \frac{\partial u}{\partial \nu}\, v \,\mathrm{d}s = 0 \tag{1.15}$$

and we can rewrite previous equation in this way :

$$\sum_{i=0}^{n-1} \int_\Omega \frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_i} \,\mathrm{d}\Omega = \int_\Omega fv \,\mathrm{d}\Omega \qquad (v = 0 \text{ in } \partial\Omega)\,. \tag{1.16}$$

Let's swap sum and integration again

$$\int_\Omega \sum_{i=0}^{n-1} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega \qquad (v = 0 \text{ in } \partial\Omega) \tag{1.17}$$

and rewrite it to operator form

$$\int_\Omega \nabla u \nabla v \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega \qquad (v = 0 \text{ in } \partial\Omega)\,. \tag{1.18}$$

So, this is weak formulation of Poisson differential equation.

Now we can use Galerkin method to discretize this equation and get rid of rest of solution's derivatives in it. But it's another part.

[Bou10][Str08a, Str08b, Str08e, Str08f]

2. The Finite Element Method (FEM)

2.1. Galerkin method

In previous chapter we derived weak formulation of the Poisson equation

$$\int_\Omega \nabla u \nabla v \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega \qquad (v = 0 \text{ in } \partial\Omega)\,. \tag{2.1}$$

Imagine now, that the system of n functions exists $\{\Phi_j\}_0^{m-1}$ (we will call them *trial functions)* defined on domain Ω. Let's put a grid with m nodes on Ω and denote this set of nodes $\{x_j\}_0^{m-1}$. Then, let's define trial function values like that:

$$\Phi_j\left(x_k\right) = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases} \qquad \wedge \qquad \sum_{i=0}^{m-1} \Phi_j\left(x\right) = 1 \quad x \in \Omega. \tag{2.2}$$

In human language it means, that every node has its own trial function, which equals one at this node and equals zero at all other nodes. At the same time sum of all trial functions has to equals one at any place in domain Ω. Not only at nodes!

If we choose system of linear trial functions it could be then set of triangels (in case of one-dimensional Ω)

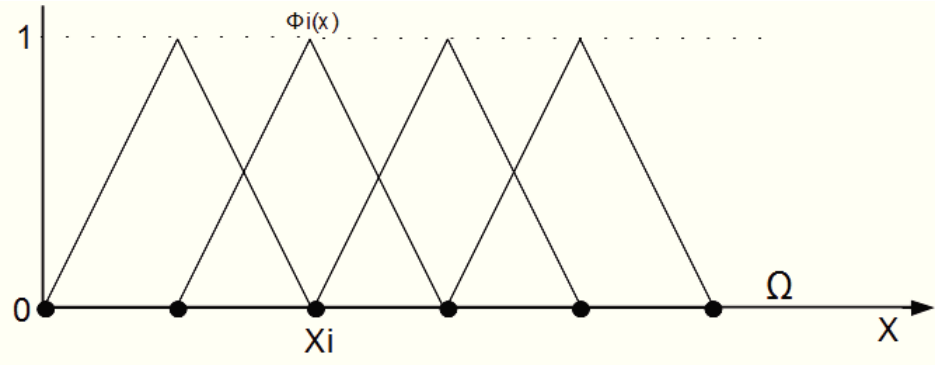


Figure 2.1.: Linear trial functions for one-dimensional case, e.g. string.

or set of the pyramids (in case of two-dimensional Ω). Nodes then will be at their peaks.

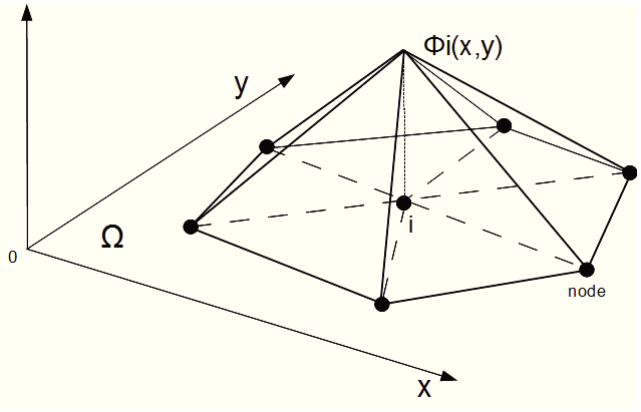


Figure 2.2.: One from the set of linear trial functions for two-dimensional case, e.g. membrane.

This system of trial functions is in fact base of vector space and we can approximately express our solution u as a vector of this space by this way :

$$u \approx \sum_{j=0}^{m-1} u_j \Phi_j, \tag{2.3}$$

where u_j are coordinates in this vector space and at the same time it's the weak solution, we are looking for.
Now, when we put it into the weak formulation, we get :

$$\int_\Omega \sum_{i=0}^{n-1} \frac{\partial}{\partial x_i} \left(\sum_{j=0}^{m-1} u_j \Phi_j \right) \frac{\partial v}{\partial x_i} \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega. \tag{2.4}$$

We can change again order of deriving and summing, integrating and summing and put u_j to the front as it is constant vector

$$\int_\Omega \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} u_j \frac{\partial \Phi_j}{\partial x_i} \frac{\partial v}{\partial x_i} \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega. \tag{2.5}$$

And reorder the equation again

$$\sum_{j=0}^{m-1} u_j \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_j}{\partial x_i} \frac{\partial v}{\partial x_i} \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega. \tag{2.6}$$

Since all trial functions Φ_k satisfy conditions for test function v, we can instead of test function v put any trial function Φ_k and then we get m equations:

$$\sum_{j=0}^{m-1} u_j \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_j}{\partial x_i} \frac{\partial \Phi_0}{\partial x_i} \,\mathrm{d}\Omega = \int_\Omega f \Phi_0 \,\mathrm{d}\Omega \tag{2.7}$$

$$\sum_{j=0}^{m-1} u_j \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_j}{\partial x_i} \frac{\partial \Phi_1}{\partial x_i} \,\mathrm{d}\Omega = \int_\Omega f \Phi_1 \,\mathrm{d}\Omega \tag{2.8}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\sum_{j=0}^{m-1} u_j \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_j}{\partial x_i} \frac{\partial \Phi_k}{\partial x_i} \,\mathrm{d}\Omega = \int_\Omega f \Phi_k \,\mathrm{d}\Omega \tag{2.9}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\sum_{j=0}^{m-1} u_j \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_j}{\partial x_i} \frac{\partial \Phi_{m-1}}{\partial x_i} \,\mathrm{d}\Omega = \int_\Omega f \Phi_{m-1} \,\mathrm{d}\Omega. \tag{2.10}$$

Let's define matrix K for n-dimensional case:

$$K = \begin{bmatrix} \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_0}{\partial x_i} \frac{\partial \Phi_0}{\partial x_i} \mathrm{d}\Omega & \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_0}{\partial x_i} \frac{\partial \Phi_1}{\partial x_i} \mathrm{d}\Omega & \cdots & \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_0}{\partial x_i} \frac{\partial \Phi_{m-1}}{\partial x_i} \mathrm{d}\Omega \\ \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_1}{\partial x_i} \frac{\partial \Phi_0}{\partial x_i} \mathrm{d}\Omega & \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_1}{\partial x_i} \frac{\partial \Phi_1}{\partial x_i} \mathrm{d}\Omega & \cdots & \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_1}{\partial x_i} \frac{\partial \Phi_{m-1}}{\partial x_i} \mathrm{d}\Omega \\ \vdots & \vdots & \ddots & \vdots \\ \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_{m-1}}{\partial x_i} \frac{\partial \Phi_0}{\partial x_i} \mathrm{d}\Omega & \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_{m-1}}{\partial x_i} \frac{\partial \Phi_1}{\partial x_i} \mathrm{d}\Omega & \cdots & \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_{m-1}}{\partial x_i} \frac{\partial \Phi_{m-1}}{\partial x_i} \mathrm{d}\Omega \end{bmatrix} \tag{2.11}$$

and vectors u, f

$$u = \begin{bmatrix} u_0 & u_1 & \cdots & u_{m-1} \end{bmatrix}^T \qquad f = \begin{bmatrix} \int_\Omega f \Phi_0 \mathrm{d}\Omega & \int_\Omega f \Phi_1 \mathrm{d}\Omega & \cdots & \int_\Omega f \Phi_{m-1} \mathrm{d}\Omega \end{bmatrix}^T \tag{2.12}$$

Then we can rewrite our system of equations as :

$$Ku = f, \tag{2.13}$$

where matrix K is called stiffness matrix, vector u displacement vector and vector f load vector.

Of course, we need to specify boundary condition at certain node(s). Let's say, we want to force some u_k to be zero. Then it's quite easy to adjust K and f in proper way. We just zero k-th row and k-th column of matrix K and instead of $K_{k,k}$ insert number 1.

$$\begin{bmatrix} & & & & 0 & & & & \\ & & & & \vdots & & & & \\ & & & & 0 & & & & \\ & & & & 0 & & & & \\ 0 & \cdots & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ & & & & 0 & & & & \\ & & & & 0 & & & & \\ & & & & \vdots & & & & \\ & & & & \vdots & & & & \\ & & & & \vdots & & & & \\ & & & & 0 & & & & \end{bmatrix} \begin{bmatrix} u_0 \\ \vdots \\ \vdots \\ u_{k-1} \\ u_k \\ u_{k+1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_{m-1} \end{bmatrix} = \begin{bmatrix} f_0 \\ \vdots \\ \vdots \\ f_{k-1} \\ 0 \\ f_{k+1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_{m-1} \end{bmatrix} \tag{2.14}$$

This equation can be now easily solved e.g. by iterative method (conjugate gradient (CG) method).

[Bou10, Str08a, Str08b, Str08e, Str08f]

2.2. Example in 2-D

Let's show, how to assemble stiffness matrix and load vector for simple 2-D example.

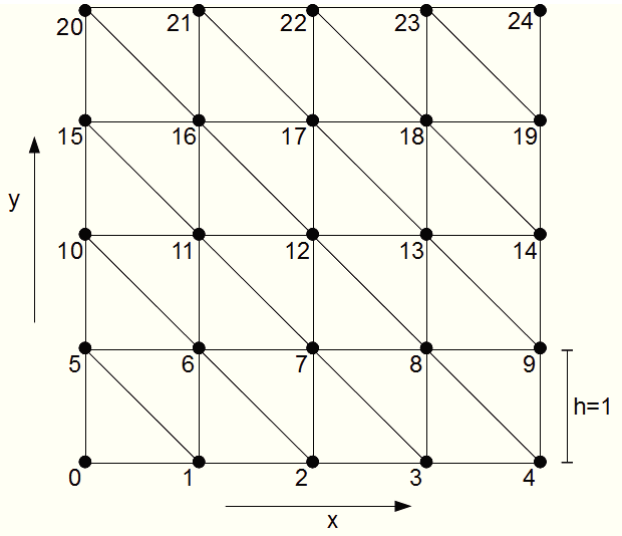


Figure 2.3.: The membrane with mesh - 2-D example.

Pyramid Φ_0 is over triangle 0-1-5 and it's $\Phi_0 = -\frac{x}{h} - \frac{y}{h} + 1$ and because $h = 1$ it's $\Phi_0 = -x - y + 1$.
Now we can derive:

$$K_{0,0} = \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_0}{\partial x_i} \frac{\partial \Phi_0}{\partial x_i} \mathrm{d}\Omega = \int_0^1 \int_0^{1-x} \left((-1)(-1) + (-1)(-1)\right) dydx = 1. \quad (2.15)$$

Pyramid Φ_1 is over triangles 0-1-5, 1-5-6 and 1-2-6. Since triangle 0-1-5 is the only common domain for Φ_0 and Φ_1, to compute $K_{0,1}$ we will need only part over 0-1-5 and it's $\Phi_{1,0-1-5} = x$.

$$K_{0,1} = \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_0}{\partial x_i} \frac{\partial \Phi_1}{\partial x_i} \mathrm{d}\Omega = \int_0^1 \int_0^{1-x} \left((-1)\,1 + (-1)\,0\right) dydx = -\frac{1}{2}. \quad (2.16)$$

Because multiplication is commutative it's obvious that $K_{1,0} = K_{0,1}$ and that's the reason why stiffness matrix is symmetric.
Another non-zero member on 0-th row will be $K_{0,5}$. It's the similar situation as with $K_{0,1}$but $\Phi_{5,0-1-5} = y$. Then

$$K_{0,5} = \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_0}{\partial x_i} \frac{\partial \Phi_5}{\partial x_i} \mathrm{d}\Omega = \int_0^1 \int_0^{1-x} \left((-1)\,0 + (-1)\,1\right) dydx = -\frac{1}{2}. \quad (2.17)$$

We could expect that, because our example is symmetric according to the line $y = x$.
All other members on 0-th row will be zeros, because there are no other pyramids,

that have common domain with Φ_0.
Now, let's compute 0-th member of f. To make it simple as much as possible let $f = 1$. Then

$$f_0 = \int_\Omega f\Phi_0 \mathrm{d}\Omega = \int_0^1 \int_0^{1-x} (-x - y + 1)\, dydx = \frac{1}{6}. \tag{2.18}$$

Another important element is

$$K_{1,5} = K_{5,1} = \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_1}{\partial x_i} \frac{\partial \Phi_5}{\partial x_i} \mathrm{d}\Omega = \int_0^1 \int_0^{1-x} ((1)\, 0 + 0\, (1))\, dydx = 0 \tag{2.19}$$

and

$$K_{1,1} = \int_\Omega \sum_{i=0}^{n-1} \frac{\partial \Phi_1}{\partial x_i} \frac{\partial \Phi_1}{\partial x_i} \mathrm{d}\Omega = 2. \tag{2.20}$$

All other members of K and f can be computed by combination of values already computed above.

Now, we can assembly our equation $Ku = f$

$$\begin{bmatrix}
1 & -\frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & \cdots \\
-\frac{1}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & \cdots \\
0 & -\frac{1}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & \cdots \\
0 & 0 & -\frac{1}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & \cdots \\
0 & 0 & 0 & -\frac{1}{2} & 1 & 0 & 0 & 0 & 0 & \cdots \\
-\frac{1}{2} & 0 & 0 & 0 & 0 & 2 & -1 & 0 & 0 & \cdots \\
0 & -\frac{1}{2} & 0 & 0 & 0 & -1 & 4 & -1 & 0 & \cdots \\
0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & -1 & 4 & -1 & \cdots \\
0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & -1 & 4 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ \vdots \end{bmatrix}
=
\begin{bmatrix} \frac{1}{6} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{2} \\ 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix}. \tag{2.21}$$

2.3. Program in PETSc and graphical solution

Here is the sample of my programming code in C language using PETSc library, which solves FEM and graphical solution of the example from the previous section

```
void FEM(Mat Ks, Vec fs,PetscBool *lsa, PetscInt vSize, PetscInt hSize)
{
  PetscInt i,j;
  for(i=0;i<(vSize-1);i++){
    for(j=0;j<(hSize-1);j++){
      if((*(lsa+i*hSize+j))&&(*(lsa+i*hSize+j+1))&&
         (*(lsa+(i+1)*hSize+j))&&(*(lsa+(i+1)*hSize+j+1))){
        Add2Triangles(Ks,fs,i*hSize+j,i*hSize+j+1,(i+1)*hSize+j,(i+1)*hSize+j+1);
      }
    }
  }
}

void Add2Triangles(Mat Ks, Vec fs, PetscInt a, PetscInt b, PetscInt c, PetscInt d)
{
  MatSetValue(Ks,a,a,1.0,ADD_VALUES);
  MatSetValue(Ks,a,b,-0.5,ADD_VALUES);
  MatSetValue(Ks,a,c,-0.5,ADD_VALUES);
  MatSetValue(Ks,b,a,-0.5,ADD_VALUES);
  MatSetValue(Ks,b,b,0.5,ADD_VALUES);
  MatSetValue(Ks,c,a,-0.5,ADD_VALUES);
  MatSetValue(Ks,c,c,0.5,ADD_VALUES);
  VecSetValue(fs,a,1.0/6.0,ADD_VALUES);
  VecSetValue(fs,b,1.0/6.0,ADD_VALUES);
  VecSetValue(fs,c,1.0/6.0,ADD_VALUES);

  MatSetValue(Ks,d,d,1.0,ADD_VALUES);
  MatSetValue(Ks,d,b,-0.5,ADD_VALUES);
  MatSetValue(Ks,d,c,-0.5,ADD_VALUES);
  MatSetValue(Ks,b,d,-0.5,ADD_VALUES);
  MatSetValue(Ks,b,b,0.5,ADD_VALUES);
  MatSetValue(Ks,c,d,-0.5,ADD_VALUES);
  MatSetValue(Ks,c,c,0.5,ADD_VALUES);
  VecSetValue(fs,d,1.0/6.0,ADD_VALUES);
  VecSetValue(fs,b,1.0/6.0,ADD_VALUES);
  VecSetValue(fs,c,1.0/6.0,ADD_VALUES);
}

void DirichletBoundaryCondition(Mat Ks, Vec fs, PetscInt n, PetscReal value)
{
  PetscInt i,nRowK,nColK;
  MatGetSize(Ks,&nRowK,&nColK);
  MatAssemblyBegin(Ks,MAT_FLUSH_ASSEMBLY);
  MatAssemblyEnd(Ks,MAT_FLUSH_ASSEMBLY);
  for(i=0;i<nRowK;i++) MatSetValue(Ks,n,i,0.0,INSERT_VALUES);
  for(i=0;i<nColK;i++) MatSetValue(Ks,i,n,0.0,INSERT_VALUES);
  MatSetValue(Ks,n,n,1.0,INSERT_VALUES);
  VecSetValue(fs,n,value,INSERT_VALUES);
}
```

As you can see in following picture, I put Dirichlet boundary condition on left and bottom side. To create this visualisation i used function PetscDrawTensorContour() from PETSc library.

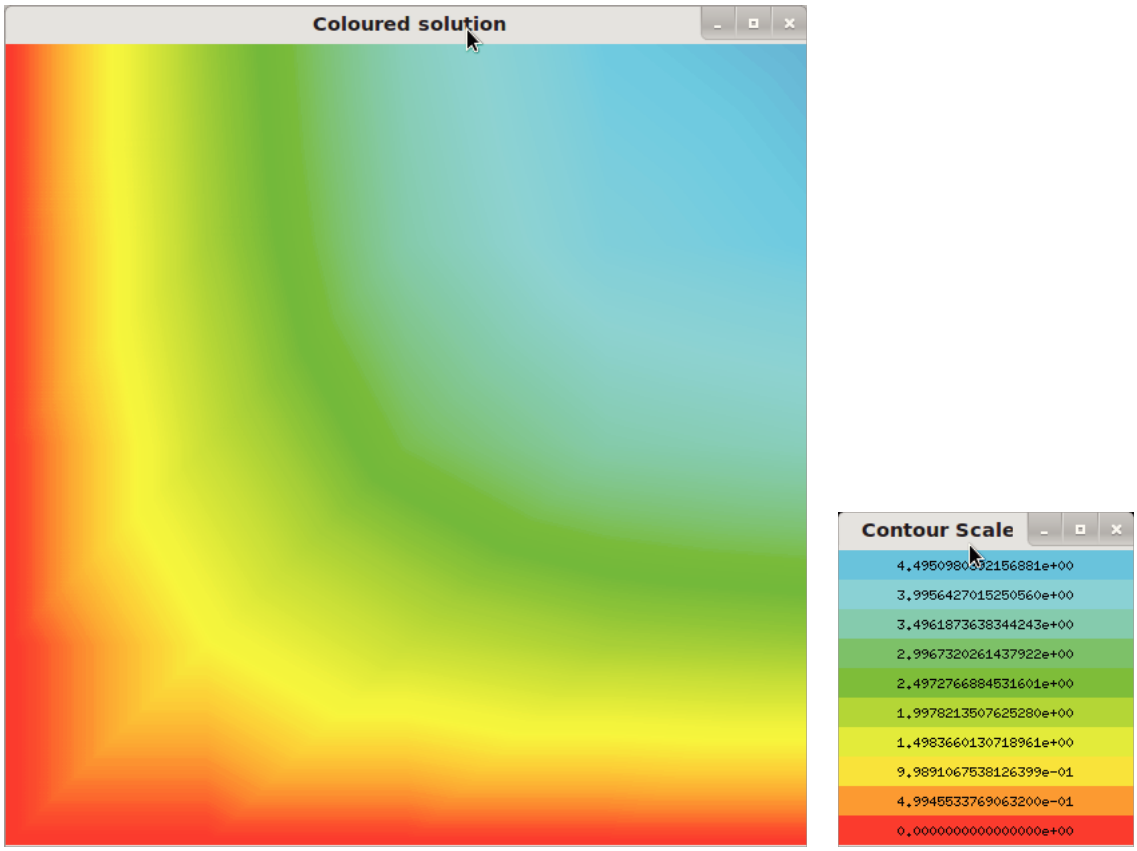


Figure 2.4.: Visualisation of the solution of example from 2.2.

3. The Finite Element Tearing and Interconnecting Method (FETI-DP)

3.1. Introduction

As you could see in previous chapter, stiffness matrix reaches really huge dimensions (squared number of nodes). Even fast computers would need enormous amount of time and memory to solve it. But present computers and mainly supercomputers can run programmes on many processors simultaneously. New computing methods are taking advantage of that. They divide main domain into the subdomains and solve them in parallel, saving time in this way. That's why we call them domain decomposition methods. There are many types of them. Most famous are FETI-1, FETI-2, FETI-C, TFETI-1 (proposed by prof. Dostal from our department :-) and finally FETI-DP, what I will focus on in this work.

3.2. How does it work

FETI-DP method, as well as other domain decomposition methods, divides (tears) main domain into the local subdomains, solves associated local problems and assemble the global solution from local ones. Situation could look like a following potato shaped example :

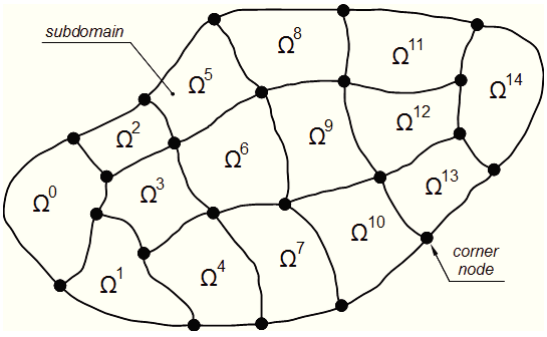


Figure 3.1.: Main domain, subdomains and corners.

FETI-DP is based on a dual-primal formulation of the FETI-2 concept and replaces the one- and two-level static and transient FETI algorithms by a single dual-primal FETI-DP method.
Let K^s, u^s and f^s denote each subdomain's stiffness matrix, displacement and load vector. We can reorganize them in following way :

$$K^s = \begin{bmatrix} K^s_{ii} & K^s_{ib} \\ K^{s^T}_{ib} & K^s_{bb} \end{bmatrix} \qquad u^s = \begin{bmatrix} u^s_i \\ u^s_b \end{bmatrix} \qquad f^s = \begin{bmatrix} f^s_i \\ f^s_b \end{bmatrix}, \tag{3.1}$$

where the subscripts i and b designate the subdomain internal and interface boundary nodes. We furthermore partition the component u^s_b as follows:

$$u^s_b = \begin{bmatrix} u^s_{br} \\ u^s_{bc} \end{bmatrix}, \tag{3.2}$$

where the additional subscript c designates nodes attached to some corners of the mesh decomposition, and the additional subscript r designates the nodes, that are boundary nodes but not attached to any corner. We call them remainders.
Here, we can choose corner's choosing strategy. Since we can take any remainder as a corner, a lot of variations arising herefrom. I will try to find ideal ratio of remainders to corners in chapter Numerical experiments.
The continuity of the displacement field at the subdomain interfaces can be written as:

$$u^s_b = u^t_b \quad \Longrightarrow \quad u^s_b - u^t_b = 0 \qquad \text{on } \Omega^s \cap \Omega^t \tag{3.3}$$

or

$$\sum_{s=0}^{N^s-1} B^s u^s = 0, \tag{3.4}$$

where B^s is a signed boolean matrix defined by

$$B^s u^s = \pm u^s_b \tag{3.5}$$

and the sign of this equality is determined by a suitable convention.

Following the FETI methodology, we seek to introduce Lagrange multipliers λ for enforcing the continuity condition, and employ a CG algorithm for determining the values of these multipliers. Now, let's re-order K^s, u^s and f^s as

$$K^s = \begin{bmatrix} K^s_{rr} & K^s_{rc} \\ K^{s^T}_{rc} & K^s_{cc} \end{bmatrix} \qquad u^s = \begin{bmatrix} u^s_r \\ u^s_{bc} \end{bmatrix} \qquad f^s = \begin{bmatrix} f^s_r \\ f^s_{bc} \end{bmatrix}, \tag{3.6}$$

where

$$u^s_r = \begin{bmatrix} u^s_i \\ u^s_{br} \end{bmatrix} \qquad \text{and} \qquad f^s_r = \begin{bmatrix} u^s_i \\ u^s_{br} \end{bmatrix} \tag{3.7}$$

and introduce the global vector of corner nodes

$$u_c = \begin{bmatrix} u^0_c & \cdots & u^k_c & \cdots & u^{N_c-1}_c \end{bmatrix}^T, \tag{3.8}$$

where u^k_c denotes a subset or all of the displacement d.o.f. attached to the k-th global node that is also a corner node of the mesh decomposition, and N_c denotes the total number of selected corner nodes. For each subdomain Ω^s, we also define two additional boolean matrices B^s_r and B^s_c by

$$B^s_r u^s_r = \pm u^s_{br} \qquad \textit{and} \qquad B^s_c u_c = u^s_{bc} \tag{3.9}$$

Using previous equations, the subdomain equations of equilibrium can be written as

$$K^s_{rr} u^s_r + K^s_{rc} B^s_c u_c = f^s_r - B^{s^T}_r \lambda \tag{3.10}$$

$$\sum_{s=0}^{N^s-1} B^{s^T}_c K^{s^T}_{rc} u^s_r + \sum_{s=0}^{N^s-1} B^{s^T}_c K^{s^T}_{rc} B^s_c u_c = \sum_{s=0}^{N^s-1} B^{s^T}_c f^s_{bc} = f_c \tag{3.11}$$

and the interface continuity condition can be re-written as

$$\sum_{s=0}^{N^s-1} B^s_c u^s_r = 0. \tag{3.12}$$

Now, let's define global matrix

$$K_{cc} = \sum_{s=0}^{N^s-1} B_c^{s^T} K_{cc}^s B_c^s \tag{3.13}$$

and from equations before we can derive, that

$$u_r^s = K_{rr}^{s^{-1}} \left(f_r^s - B_r^{s^T} \lambda - K_{rc}^s B_c^s u_c \right). \tag{3.14}$$

Substituting previous equations leads after some algebraic transformations to

$$\begin{bmatrix} F_{I_{rr}} & F_{I_{rc}} \\ F_{I_{rc}}^T & -K_{cc}^* \end{bmatrix} \begin{bmatrix} \lambda \\ u_c \end{bmatrix} = \begin{bmatrix} d_r \\ -f_c^* \end{bmatrix}. \tag{3.15}$$

Here I'd like to warn about probably typo on page 1528 in [CF01]. There is f_c^* and should be $-f_c^*$. Minus sign is missing ;-).

$$F_{I_{rr}} = \sum_{s=0}^{N^s-1} B_r^s K_{rr}^{s^{-1}} B_r^{s^T} \tag{3.16}$$

$$F_{I_{rc}} = \sum_{s=0}^{N^s-1} B_r^s K_{rr}^{s^{-1}} K_{rc}^s B_c^s \tag{3.17}$$

$$K_{cc}^* = K_{cc} - \sum_{s=0}^{N^s-1} \left(K_{rc}^s B_c^s \right)^T K_{rr}^{s^{-1}} K_{rc}^s B_c^s \tag{3.18}$$

$$d_r = \sum_{s=0}^{N^s-1} B_r^s K_{rr}^{s^{-1}} f_r^s \tag{3.19}$$

$$f_c^* = f_c - \sum_{s=0}^{N^s-1} B_c^{s^T} K_{rc}^{s^T} K_{rr}^{s^{-1}} f_r \tag{3.20}$$

The above problem is a dual-primal problem as it relates the dual Lagrange multiplier unknowns λ to the primal displacement d.o.f. u_c. By eliminating u_c, it can transformed however into the following symmetric positive definitive dual interface

problem

$$\left(F_{I_{rr}} + F_{I_{rc}} K_{cc}^{*^{-1}} F_{I_{rc}}^T\right) \lambda = d_r - F_{I_{rc}} K_{cc}^{*^{-1}} f_c^* \tag{3.21}$$

and then

$$u_c = K_{cc}^{*^{-1}} \left(F_{I_{rc}}^T \lambda + f_c^*\right) \tag{3.22}$$

$$u_r^s = K_{rr}^{s^{-1}} \left(f_r^s - B_r^{s^T} \lambda - K_{rc}^s B_c^s u_c\right). \tag{3.23}$$

[CF01][Hor07, Dos05, Dos06]

3.3. Example in 2-D

There is no better way to explain something than show a simple example. Let's take then example from chapter 2 and apply FETI-DP method on it. When we tear it into 4 subdomains we get this

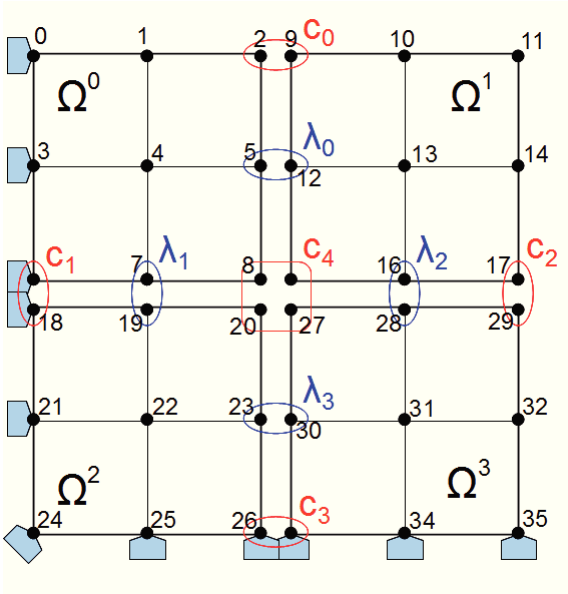


Figure 3.2.: Example from section 2.2 teared into 4 subdomains.

$$u_r^0 = \begin{bmatrix} u_0 \\ u_1 \\ u_3 \\ u_4 \\ u_5 \\ u_7 \end{bmatrix} \quad u_r^1 = \begin{bmatrix} u_{10} \\ u_{11} \\ u_{13} \\ u_{14} \\ u_{12} \\ u_{16} \end{bmatrix} \quad u_r^2 = \begin{bmatrix} u_{21} \\ u_{22} \\ u_{24} \\ u_{25} \\ u_{19} \\ u_{23} \end{bmatrix} \quad u_r^3 = \begin{bmatrix} u_{31} \\ u_{32} \\ u_{34} \\ u_{35} \\ u_{28} \\ u_{30} \end{bmatrix}$$

$$u_{bc}^0 = \begin{bmatrix} u_2 \\ u_6 \\ u_8 \end{bmatrix} \quad u_{bc}^1 = \begin{bmatrix} u_9 \\ u_{15} \\ u_{17} \end{bmatrix} \quad u_{bc}^2 = \begin{bmatrix} u_{18} \\ u_{20} \\ u_{26} \end{bmatrix} \quad u_{bc}^3 = \begin{bmatrix} u_{27} \\ u_{29} \\ u_{33} \end{bmatrix}$$

$$B_r^0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad B_r^1 = \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B_r^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad B_r^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$B_c^0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad B_c^1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$B_c^2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad B_c^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

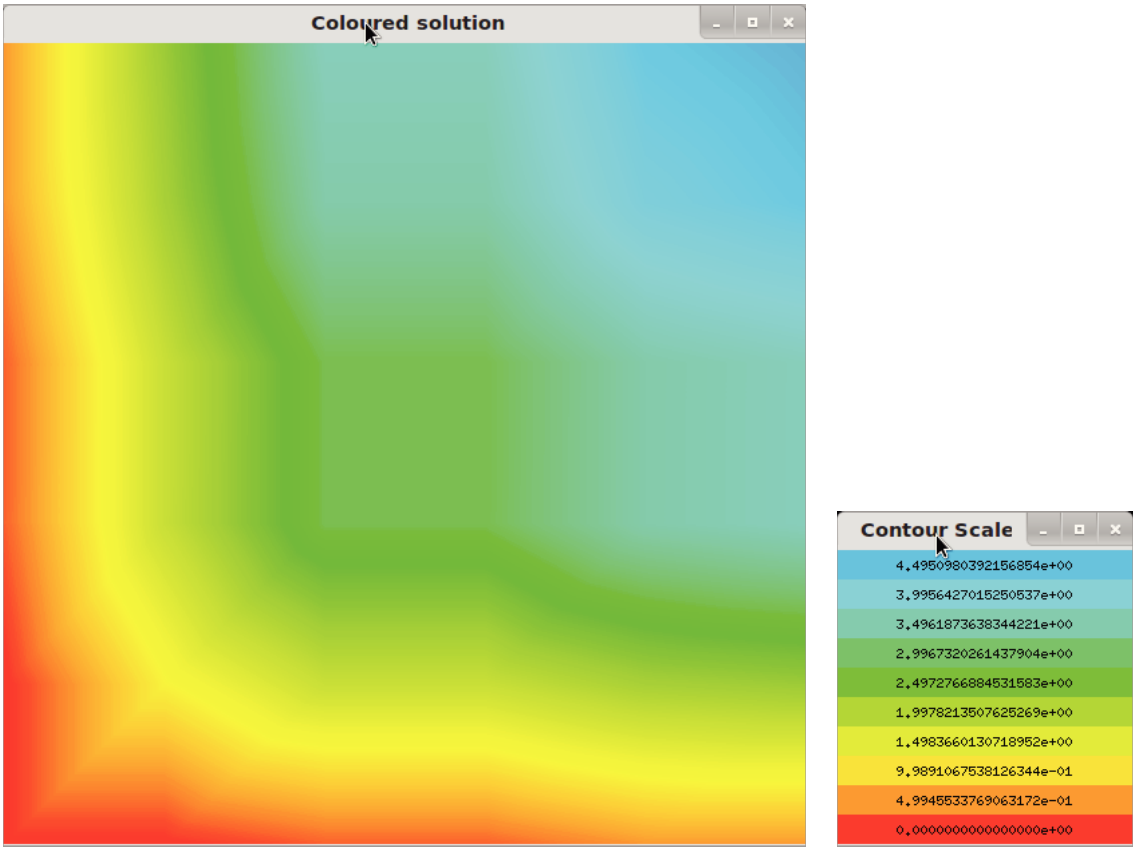


Figure 3.3.: This picture shows the solution of this example computed by my program using FETI-DP.

4. Parallel implementation of FETI-DP

4.1. PETSc

My parallel implementation is programed in pure C language using PETSc library [BBB+12]. PETSc stands for Portable Extensible Toolkit for Scientific computation.

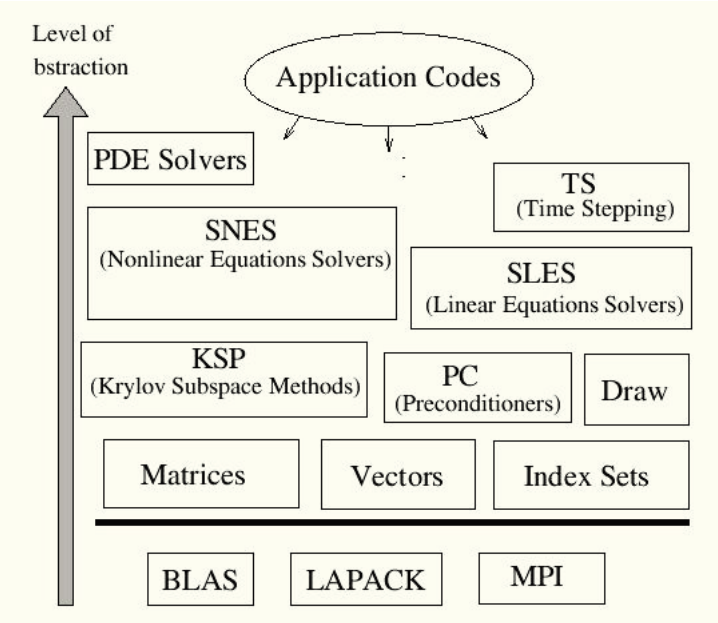


Figure 4.1.: PETSc schema.

It's a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel and sequential computers, especially used for the numerical solution of PDEs and related problems on high-performance computers. A number of parellel linear and nonlinear equation solvers and unconstrained minimization modules using modern programming paradigms enables development of large scientific codes written in C, C++ or Fortran. PETSc uses the MPI standard for all message passing communication and routines BLAS, LAPACK, MINPACK and SPARSPAK. Technique of object oriented programming provides enormous flexibility and code reuse. The library is hierarchically organized according to level of abstraction. PETSc consists of a variety of components, which

manipulate a particular family of objects. These components are index sets, vectors, matrices (sparse and dense), distributed arrays (useful for parallelizing regular grid-based problems), Krylov subspace methods, preconditioners, nonlinear solvers, unconstrained minimization, timesteppers for solving time dependent PDEs, graphic devices, etc.
Each of these components consists of an abstract interface and one or more implementations using particular data structures. Thus PETSc provides clean and effective codes for various phases of solving PDEs, with a uniform approach for each class and effective codes for various phases of solving PDEs, with a uniform approach for each class of problems, as well as a rich enviroment for modeling scientific applications and for algorithm design and prototyping.

4.2. Algorithm

My program is designed to run in parallel on the same number of processors as number of subdomains is. Thus every subdomain is solved by its own processor core. Figure 4.2 shows the algorithm of whole program.

Transition between sequential and parallel processing is realized by scattering. I used functions VecScatterCreateToAll() + VecScatterBegin() + VecScatterEnd(), which copy (scatters) data from global parallel vector to local sequential vector on each processor. And also VecScatterCreate() + VecScatterBegin() + VecScatterEnd() with option ADD_VALUES, which sum local sequential vector from all processes and save result to the global parallel vector.

I used conjugate gradient method to solve dual interface problem. It allows me not to assembly global matrix $F_{I_{rr}}$, which is really time demanding, because we need to multiply $B_r^s K_{rr}^{s^{-1}} B_r^{s^T}$ on each processor to get it. Instead of that program just multiply $B_r^s \left(K_{rr}^{s^{-1}} \left(B_r^{s^T} grad \right) \right)$ in every CG step. Here is a part of my program, solving this dual problem.

```
ierr = VecSet(lambda, 0.0);CHKERRQ(ierr); // lambda=0
ierr = VecSet(g, 0.0);CHKERRQ(ierr);      // g=A*lambda
ierr = VecAYPX(g,-1.0,rhs);CHKERRQ(ierr); // g=rhs-g (rhs = right hand side)
ierr = VecCopy(g, p);CHKERRQ(ierr);       // p=g
ierr = VecDot(g,g,&g2old);CHKERRQ(ierr);  // g2old=(g,g)

while (g2old > 0.0000000001)
{
  ierr = VecScatterCreateToAll(p,&vecscatToAll,&plocal);CHKERRQ(ierr);
  ierr = VecScatterBegin(vecscatToAll,p,plocal,INSERT_VALUES,SCATTER_FORWARD);
  ierr = VecScatterEnd(vecscatToAll,p,plocal,INSERT_VALUES,SCATTER_FORWARD);
  ierr = MatMultTranspose(Br,plocal,tempVec1);CHKERRQ(ierr);
  ierr = MatMult(Krrinv,tempVec1,tempVec2);CHKERRQ(ierr);
  ierr = MatMult(Br,tempVec2,tempVec3);CHKERRQ(ierr);
  ierr = VecZeroEntries(tempVec4);CHKERRQ(ierr);
  ierr = VecScatterBegin(vecscatNlambda,tempVec3,tempVec4,ADD_VALUES,SCATTER_FORWARD);
  ierr = VecScatterEnd(vecscatNlambda,tempVec3,tempVec4,ADD_VALUES,SCATTER_FORWARD);
  ierr = MatMultAdd(FKF,p,tempVec4,Ap);CHKERRQ(ierr);
  ierr = VecScatterDestroy(&vecscatToAll);CHKERRQ(ierr);
  ierr = VecDestroy(&plocal);CHKERRQ(ierr);
  ierr = VecDot(p,Ap,&pAp);CHKERRQ(ierr);          // pAp=p'*Ap
  alpha=g2old/pAp;                                  // alpha=g2old/pAp
  ierr = VecAXPY(lambda,alpha,p);CHKERRQ(ierr);     // lambda=lambda+alpha*p
  ierr = VecAXPY(g,-1.0*alpha,Ap);CHKERRQ(ierr);    // g=g-alpha*Ap
  ierr = VecDot(g,g,&g2new);CHKERRQ(ierr);          // g2new=(g,g)
  ierr = VecAYPX(p,g2new/g2old,g);CHKERRQ(ierr);    // p=g+(g2new/g2old)*p
  g2old=g2new;                                      // g2old=g2new
}
```

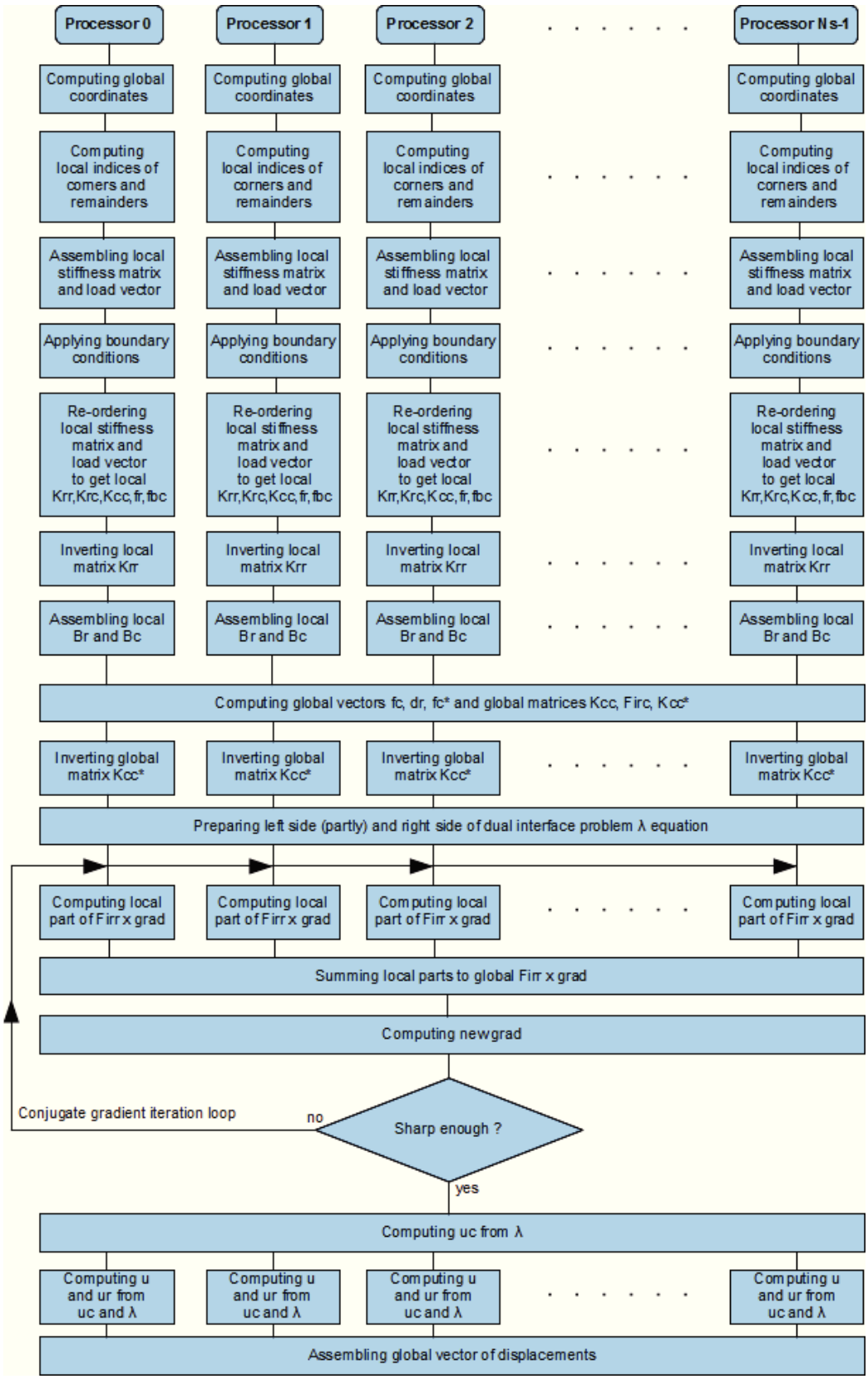


Figure 4.2.: Algorithm of my program.

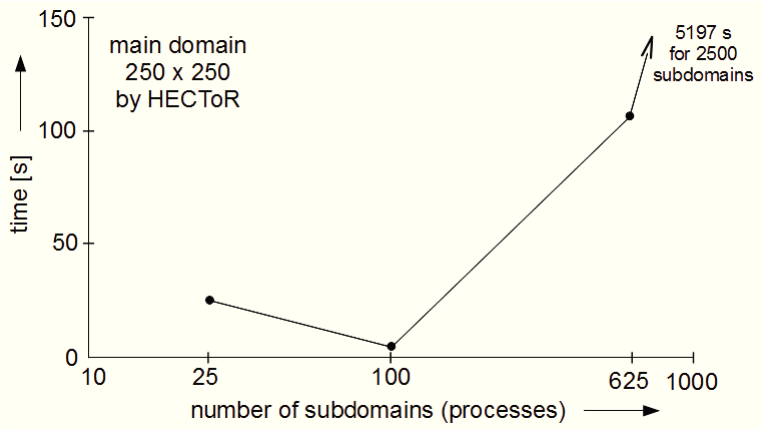
5. Numerical experiments

5.1. The Supercomputer

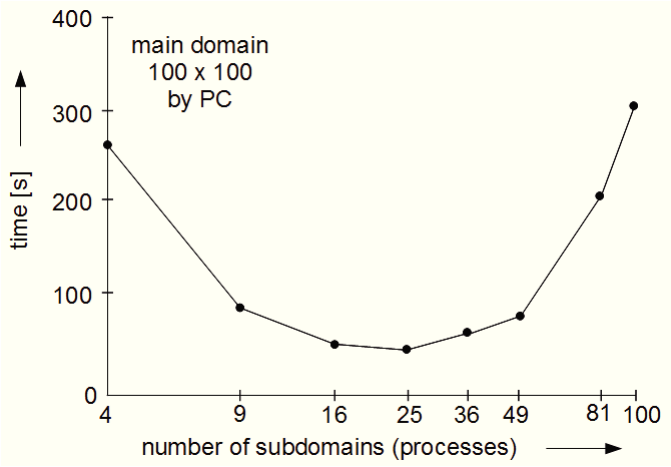
For experiments with my benchmarks I used UK's largest, fastest and most powerful supercomputer - HECToR (High End Computing Terascale Resource). It is capable of over 800 million million calculations a second - that's over 114,000 calculations a second for every man, woman and child on Earth. HECToR occupies an area of two tennis courts and has a memory of 90 Terabytes - equivalent to over 180,000 iPhone's. It also has one Petabyte of disk space for storing data. HECToR phase 3 uses the latest "Bulldozer" multicore processor architecture from AMD. This current Phase 3 system (XE6) is contained in 30 cabinets and comprise of a total of 704 compute blades. Each blade contains four compute nodes giving a total of 2816 compute nodes, each with two 16-core AMD Opteron 2.3GHz Interlagos processors. This amounts to a total of 90,112 cores. Each 16-core socket is coupled with a Cray Gemini routing and communications chip. Each 16-core processor shares 16Gb of memory, giving a system total of around 90 Tb. The theoretical peak performance of the phase 3 system is over 800 Tflops. There are 16 service blades on phase 3, each with two dual-core processor sockets. They act as login nodes, controllers for the I/O and for the network. There is one Gemini router chip for every two XE node. This Gemini chip has 10 network links which are used to implement a 3D-torus of processors. The MPI point-to-point bandwidth is 5 GB/s or more. The latency between two nodes is around 1-1.5μs.

5.2. Scalability experiments

In present time I have 4 benchmark results from HECToR. There will be more in the future.

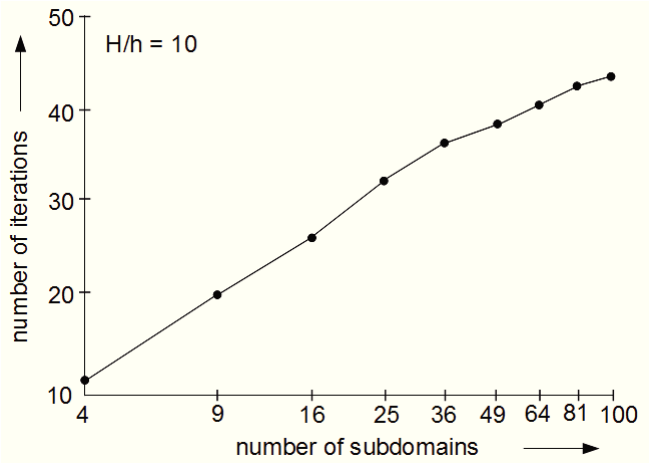


I also experimented with benchmarks on 2-cores PC.



One of the goal of this work is to prove parallel scalability method FETI-DP. Parallel scalability of some algorithm means, that run-time of this algorithm is somehow related to the number of processes it use. From both graphs can be seen, that there is a ideal combination of N^s - number of subdomains (processes) and N - main domain size. If N^s is too small, matrices K_{rr}^s are too big and then solving coarse problem takes big amount of time. On the other hand, if N^s is too big, subdomains are too small, vector λ of Lagrange multipliers has big dimension and then solving dual problem is really time demanding. Therefore, we can confirm, that method FETI-DP is parallely scalable. In PETSc could be discovered also parallel efficiency (and lot of other informations) from program's output when -log_summary option is used. There is Max/Min column that represents ratio maximum to minimum recources using by processes. It should be equal to one as close as possible. See HECToR's output in the appendix to check parallel efficiency of my program.

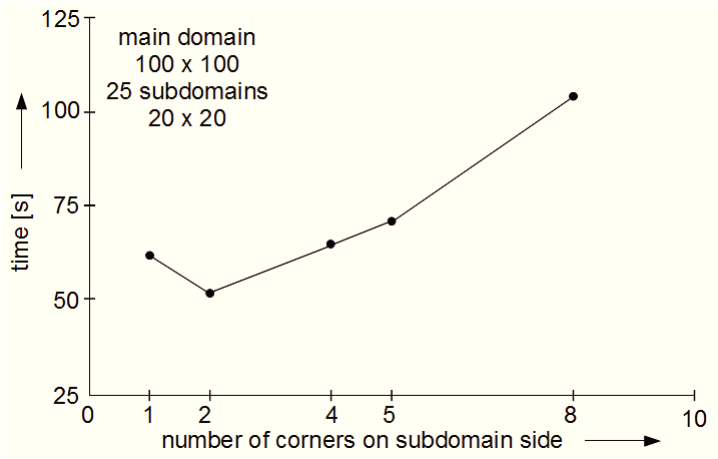
Numerical scalability is another algorithm attribute: that the computational resources required to solve a problem of a given size are proportionaly related to the problem size. In present time I do not have enough results from HECToR supercomputer to prove it, so results are from PC. I set up fixed ratio $\frac{H}{h} = 10$. H is size of subdomain's side and h is mesh parameter.

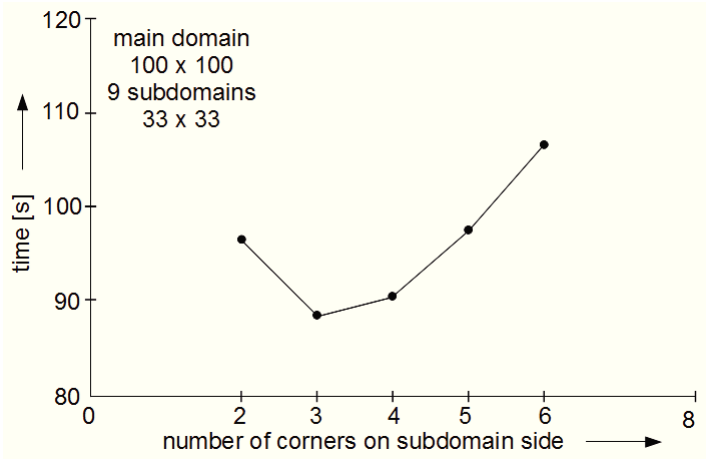


Since number of iterations from 36 subdomains is almost not changing while number of d.o.f. is changing by thousands, I can confirm, that numerical scalability has been proved.

5.3. Corner choosing strategy experiments

Here I have replaced some of remainder nodes by corners. Unfortunately, I do not have enough results from HECToR, so following graph is from PC only. Again, there will be more in the future.





As can be seen in these graphs, sometime it is possible to improve run-time by replacing some remainders by corners. For example in case of main domain teared into 9 subdomains of size 33 x 33, replacing one remainder by corner on each subdomain side, reduced run-time by 10%.

6. Conclusion and future works

In this work, I have reached all four assignment goals. I have understood FETI-DP method, learned how to use PETSc, implemented FETI-DP method in parallel using PETSc and in chapter 5 also proved the algorithm's parallel and numerical scalability by means of numerical experiments. Furthermore I have additionally explained basics of Galerkin method and finite element method in chapters 1 and 2. This paper could be also useful then for someone, who is trying to understand how to numerically solve partial differential equations.

My contribution to FETI-DP method is in the section 5.3, where I have tested something that could be called a *corner choosing strategy* in the future :-). As it could be seen in graphs, replacing some remainder nodes by corner nodes has effect on algorithm's run-time. In near future I would like to focus on reducing FETI-DP algorithm's run-time by changing number and size of subdomains and on corner choosing strategy.

Acknowledgments

I'd like to thank my wife Lucie for her understanding and Dr.Horak for his patience and english grammar check.

A. Appendix

A.1. HECToR's output

Here can be seen, how my program, running on 100 HECToR's cores, solved 250 x 250 domain (62500 d.o.f.) in just 6.44 seconds.

```
---------------------------------------------------------------------------
*** pr2c0008   Job: 1399217.sdb   starts: 23/04/13 20:49:12   host: phase3 ***
*** pr2c0008   Job: 1399217.sdb   starts: 23/04/13 20:49:12   host: phase3 ***
*** pr2c0008   Job: 1399217.sdb   starts: 23/04/13 20:49:12   host: phase3 ***
*** pr2c0008   Job: 1399217.sdb   starts: 23/04/13 20:49:12   host: phase3 ***
User may access requested budget
Computing local stiffness matrices Ks:  done.
Inverting local matrices Krr:  done.
Creating matrices Br:  done.
Creating matrices Bc:  done.
Computing global vector fc:  done.
Computing global matrix Kcc:  done.
Computing global matrix FIrc:  done.
Computing global matrix Kcc*:  done.
Computing global vector dr:  done.
Computing global vector fc*:  done.
Inverting global matrix Kcc*:  done.
Preparing lambda equation:  done.
Solving lambda equation:  done.
Computing uc from lambda:  done.
Computing local vectors ur from uc and lambda:  done.
Creating local vectors u:  done.
Creating global vector of displacements:  done.
ALL DONE.

------------------------------------------ PETSc Performance Summary: ------------------------------------------
FETI-DP_Tomcala_100x25x25 on a  named nid00985 with 100 processors, by Unknown Tue Apr 23 19:49:42 2013
Using Petsc Release Version 3.3.0, Patch 2, Fri Jul 13 15:42:00 CDT 2012

                         Max       Max/Min        Avg      Total
 Time (sec):           6.443e+00      1.00020   6.442e+00
Objects:               7.640e+02      1.00000   7.640e+02
Flops:                 1.831e+09      1.00361   1.825e+09  1.825e+11
Flops/sec:             2.842e+08      1.00361   2.833e+08  2.833e+10
MPI Messages:          6.765e+04      1.00145   6.757e+04  6.757e+06
MPI Message Lengths:   7.232e+06      1.04276   1.036e+02  7.004e+08
MPI Reductions:        6.100e+03      1.00000

Flop counting convention: 1 flop = 1 real number operation of type (multiply/divide/add/subtract)
                            e.g., VecAXPY() for real vectors of length N --> 2N flops
                            and VecAXPY() for complex vectors of length N --> 8N flops

Summary of Stages:   --- Time ----  ----- Flops -----  --- Messages ---  -- Message Lengths --  -- Reductions --
                        Avg     %Total     Avg     %Total   counts   %Total     Avg         %Total  counts   %Total
  0:  Main Stage: 6.4419e+00 100.0%  1.8250e+11 100.0%  6.757e+06 100.0%  1.036e+02      100.0%  6.099e+03 100.0%

------------------------------------------------------------------------------------------------------------------------
See the 'Profiling' chapter of the users' manual for details on interpreting output.
Phase summary info:
   Count: number of times phase was executed
   Time and Flops: Max - maximum over all processors
                   Ratio - ratio of maximum to minimum over all processors
   Mess: number of messages sent
   Avg. len: average message length
   Reduct: number of global reductions
   Global: entire computation
   Stage: stages of a computation. Set stages with PetscLogStagePush() and PetscLogStagePop().
      %T - percent time in this phase         %f - percent flops in this phase
      %M - percent messages in this phase     %L - percent message lengths in this phase
      %R - percent reductions in this phase
   Total Mflop/s: 10e-6 * (sum of flops over all processors)/(max time over all processors)
------------------------------------------------------------------------------------------------------------------------
Event           Count      Time (sec)     Flops                             --- Global ---  --- Stage ---   Total
              Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %f %M %L %R  %T %f %M %L %R Mflop/s
------------------------------------------------------------------------------------------------------------------------
```

```
--- Event Stage 0: Main Stage

MatMult        5007 1.0 1.8174e+00 1.7 6.38e+08 1.0 1.2e+06 9.4e+00 4.3e+03 26 35 17  2 70  26 35 17  2 70 35027
MatMultAdd       79 1.0 1.2174e-01 1.6 2.75e+07 1.0 0.0e+00 0.0e+00 7.9e+01  2  1  0  0  1   2  1  0  0  1 22244
MatMultTranspose 82 1.0 9.2335e-01 2.0 4.12e+08 1.0 0.0e+00 0.0e+00 2.0e+00 14 23  0  0  0  14 23  0  0  0 44548
MatLUFactor       2 1.0 3.4713e-02 1.3 1.61e+08 1.0 0.0e+00 0.0e+00 0.0e+00  0  9  0  0  0   0  9  0  0  0 463094
MatConvert        4 1.0 4.7347e-02 1.1 0.00e+00 0.0 0.0e+00 0.0e+00 8.0e+00  1  0  0  0  0   1  0  0  0  0      0
MatScale          1 1.0 1.4067e-05 2.4 2.34e+02 2.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0    973
MatAssemblyBegin 35 1.0 8.5361e-03 1.8 0.00e+00 0.0 3.0e+04 2.6e+02 1.8e+01  0  0  0  1  0   0  0  0  1  0      0
MatAssemblyEnd   35 1.0 1.7624e-02 1.3 0.00e+00 0.0 5.9e+04 4.3e+00 6.5e+01  0  0  1  0  1   0  0  1  0  1      0
MatGetRow      8518 2.0 1.1564e-01 2.4 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  1  0  0  0  0   1  0  0  0  0      0
MatGetSubMatrice  3 1.0 3.4927e-02 4.8 0.00e+00 0.0 0.0e+00 0.0e+00 6.0e+00  0  0  0  0  0   0  0  0  0  0      0
MatZeroEntries    5 1.0 2.8466e-02 4.9 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
MatAXPY           1 1.0 4.0231e-03 1.1 0.00e+00 0.0 2.0e+04 4.3e+00 1.6e+01  0  0  0  0  0   0  0  0  0  0      0
MatTranspose      2 1.0 1.0270e-02 1.1 0.00e+00 0.0 3.0e+04 2.6e+02 1.4e+01  0  0  0  1  0   0  0  0  1  0      0
MatMatMult        8 1.0 1.1822e+00 1.3 0.00e+00 0.0 0.0e+00 0.0e+00 1.6e+01 17  0  0  0  0  17  0  0  0  0      0
MatMatSolve       2 1.0 8.7347e-02 1.2 4.84e+08 1.0 0.0e+00 0.0e+00 0.0e+00  1 26  0  0  0   1 26  0  0  0 551666
MatPtAP           2 1.0 5.9929e-01 1.1 1.08e+08 1.0 0.0e+00 0.0e+00 1.2e+01  9  6  0  0  0   9  6  0  0  0 17916
MatPtAPSymbolic   2 1.0 4.4275e-01 1.1 0.00e+00 0.0 0.0e+00 0.0e+00 1.2e+01  7  0  0  0  0   7  0  0  0  0      0
MatPtAPNumeric    2 1.0 1.5744e-01 1.2 1.08e+08 1.0 0.0e+00 0.0e+00 0.0e+00  2  6  0  0  0   2  6  0  0  0 68195
VecDot          159 1.0 4.3816e-0210.9 1.32e+04 1.0 0.0e+00 0.0e+00 1.6e+02  0  0  0  0  3   0  0  0  0  3    30
VecCopy          80 1.0 2.1482e-04 2.6 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
VecSet         5511 1.0 5.2023e-03 1.7 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
VecAXPY         160 1.0 3.1114e-04 2.5 1.45e+04 1.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0  4605
VecAYPX          83 1.0 2.0289e-04 2.5 7.34e+03 1.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0  3571
VecAssemblyBegin235 1.0 2.7164e-02 1.1 0.00e+00 0.0 2.5e+04 3.5e+01 3.5e+02  0  0  0  0  6   0  0  0  0  6      0
VecAssemblyEnd  235 1.0 2.5141e-03 5.6 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
VecScatterBegin5207 1.0 7.5248e-01 1.9 0.00e+00 0.0 6.6e+06 1.0e+02 4.5e+03 10  0 98 98 74  10  0 98 98 74      0
VecScatterEnd   667 1.0 1.7111e+00 5.3 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  7  0  0  0  0   7  0  0  0  0      0
------------------------------------------------------------------------------------------------------------------------

Memory usage is given in bytes:

Object Type          Creations   Destructions     Memory  Descendants' Mem.
Reports information only for process 0.

--- Event Stage 0: Main Stage

              Matrix    59             48     87257400     0
              Vector   263            143      3552744     0
      Vector Scatter   212            209       131244     0
           Index Set   229            223       182584     0
              Viewer     1              0            0     0
========================================================================================================================
Average time to get PetscTime(): 1.90735e-07
Average time for MPI_Barrier(): 1.55926e-05
Average time for zero size MPI_Send(): 3.24965e-06
#PETSc Option Table entries:
-log_summary
#End of PETSc Option Table entries
Compiled without FORTRAN kernels
Compiled with full precision matrices (default)
sizeof(short) 2 sizeof(int) 4 sizeof(long) 8 sizeof(void*) 8 sizeof(PetscScalar) 8 sizeof(PetscInt) 4
Configure run at:
Configure options:
Application 4402182 resources: utime ~2598s, stime ~93s
------------------------------------------------------------------------------

Resources requested: mpparch=XT,mppnppn=32,mppwidth=100,ncpus=1,place=pack,walltime=00:59:00
Resources allocated: cpupercent=0,cput=00:00:02,mem=5140kb,ncpus=1,vmem=39148kb,walltime=00:00:32

*** pr2c0008   Job: 1399217.sdb   ends: 23/04/13 20:49:43   queue: par:4n_1h ***
*** pr2c0008   Job: 1399217.sdb   ends: 23/04/13 20:49:43   queue: par:4n_1h ***
*** pr2c0008   Job: 1399217.sdb   ends: 23/04/13 20:49:43   queue: par:4n_1h ***
*** pr2c0008   Job: 1399217.sdb   ends: 23/04/13 20:49:43   queue: par:4n_1h ***
```

And here is solved the same domain, divided into 2500 subdomains, by 2500 processor cores.

```
------------------------------------------------------------------------------
*** pr2c0008   Job: 1411448.sdb   starts: 30/04/13 13:06:24   host: phase3 ***
*** pr2c0008   Job: 1411448.sdb   starts: 30/04/13 13:06:24   host: phase3 ***
*** pr2c0008   Job: 1411448.sdb   starts: 30/04/13 13:06:24   host: phase3 ***
*** pr2c0008   Job: 1411448.sdb   starts: 30/04/13 13:06:24   host: phase3 ***

User may access requested budget
Computing local stiffness matrices Ks:  done.
Inverting local matrices Krr:  done.
Creating matrices Br:  done.
Creating matrices Bc:  done.
Computing global vector fc:  done.
Computing global matrix Kcc:  done.
Computing global matrix FIrc:  done.
Computing global matrix Kcc*:  done.
Computing global vector dr:  done.
Computing global vector fc*:  done.
```

```
Inverting global matrix Kcc*:  done.
Preparing lambda equation:  done.
Solving lambda equation:  done.
Computing uc from lambda:  done.
Computing local vectors ur from uc and lambda:  done.
Creating local vectors u:  done.
Creating global vector of displacements:  done.
ALL DONE.

---------------------------------------------- PETSc Performance Summary: ----------------------------------

FETI-DP_Tomcala_2500x5x5 on a  named nid00003 with 2500 processors, by Unknown Tue Apr 30 13:33:46 2013
Using Petsc Release Version 3.3.0, Patch 2, Fri Jul 13 15:42:00 CDT 2012

                         Max       Max/Min        Avg      Total
Time (sec):           5.197e+03      1.00001   5.197e+03
Objects:              8.025e+03      1.00000   8.025e+03
Flops:                3.510e+11      1.00030   3.510e+11  8.774e+14
Flops/sec:            6.753e+07      1.00030   6.753e+07  1.688e+11
MPI Messages:         3.256e+07      1.00008   3.256e+07  8.139e+10
MPI Message Lengths:  6.276e+08      1.26175   1.611e+01  1.311e+12
MPI Reductions:       3.853e+04      1.00000

Flop counting convention: 1 flop = 1 real number operation of type (multiply/divide/add/subtract)
                            e.g., VecAXPY() for real vectors of length N --> 2N flops
                            and VecAXPY() for complex vectors of length N --> 8N flops

Summary of Stages:   ----- Time ------  ----- Flops -----  --- Messages ---  -- Message Lengths --  -- Reductions --
                        Avg     %Total     Avg     %Total   counts   %Total     Avg         %Total   counts   %Total
  0:      Main Stage: 5.1971e+03 100.0%  8.7738e+14 100.0%  8.139e+10 100.0%  1.611e+01      100.0%  3.853e+04 100.0%

------------------------------------------------------------------------------------------------------------------------
See the 'Profiling' chapter of the users' manual for details on interpreting output.
Phase summary info:
   Count: number of times phase was executed
   Time and Flops: Max - maximum over all processors
                   Ratio - ratio of maximum to minimum over all processors
   Mess: number of messages sent
   Avg. len: average message length
   Reduct: number of global reductions
   Global: entire computation
   Stage: stages of a computation. Set stages with PetscLogStagePush() and PetscLogStagePop().
      %T - percent time in this phase         %f - percent flops in this phase
      %M - percent messages in this phase     %L - percent message lengths in this phase
      %R - percent reductions in this phase
   Total Mflop/s: 10e-6 * (sum of flops over all processors)/(max time over all processors)
------------------------------------------------------------------------------------------------------------------------
Event                Count      Time (sec)     Flops                             --- Global ---  --- Stage ---
Total
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %f %M %L %R  %T %f %M %L %R Mflop/s
------------------------------------------------------------------------------------------------------------------------

--- Event Stage 0: Main Stage

MatMult       30339 1.0 1.6662e+03 1.1 3.04e+11 1.0 1.6e+10 8.3e+00 1.7e+04 31 87 20 10 45  31 87 20 10 45 455963
MatMultAdd       25 1.0 3.1265e+0020.2 4.41e+06 1.2 0.0e+00 0.0e+00 2.5e+01  0  0  0  0  0   0  0  0  0  0  3456
MatMultTranspose 28 1.0 5.2992e-02 1.8 1.69e+07 1.0 0.0e+00 0.0e+00 2.0e+00  0  0  0  0  0   0  0  0  0  0 759516
MatLUFactor       2 1.0 1.6613e+00 1.1 1.17e+10 1.0 0.0e+00 0.0e+00 0.0e+00  0  3  0  0  0   0  3  0  0  0 17571652
MatConvert        4 1.0 8.2247e-03 1.2 0.00e+00 0.0 0.0e+00 0.0e+00 8.0e+00  0  0  0  0  0   0  0  0  0  0      0
MatScale          1 1.0 6.5088e-05 5.9 5.19e+03 2.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0 103619
MatAssemblyBegin 33 1.0 1.1569e+00 2.9 0.00e+00 0.0 1.9e+07 3.4e+01 1.6e+01  0  0  0  0  0   0  0  0  0  0      0
MatAssemblyEnd   33 1.0 1.4262e+00 1.1 0.00e+00 0.0 3.7e+07 4.1e+00 5.6e+01  0  0  0  0  0   0  0  0  0  0      0
MatGetRow     34598 2.0 1.4738e+00 2.5 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
MatGetSubMatrice  3 1.0 1.7822e-02105.6 0.00e+00 0.0 0.0e+00 0.0e+00 6.0e+00  0  0  0  0  0   0  0  0  0  0
0
MatZeroEntries    5 1.0 3.1876e-02 1.9 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
MatAXPY           1 1.0 2.8499e-01 1.3 0.00e+00 0.0 1.2e+07 4.1e+00 1.6e+01  0  0  0  0  0   0  0  0  0  0      0
MatTranspose      2 1.0 6.6490e-01 1.0 0.00e+00 0.0 1.9e+07 3.4e+01 1.4e+01  0  0  0  0  0   0  0  0  0  0      0
MatMatMult        8 1.0 4.3745e+00 5.5 0.00e+00 0.0 0.0e+00 0.0e+00 1.6e+01  0  0  0  0  0   0  0  0  0  0      0
MatMatSolve       2 1.0 4.7298e+00 1.1 3.50e+10 1.0 0.0e+00 0.0e+00 0.0e+00  0 10  0  0  0   0 10  0  0  0 18512207
MatPtAP           2 1.0 4.6453e+00 1.8 3.40e+08 1.0 0.0e+00 0.0e+00 1.2e+01  0  0  0  0  0   0  0  0  0  0 182762
MatPtAPSymbolic   2 1.0 2.6485e+00 1.7 0.00e+00 0.0 0.0e+00 0.0e+00 1.2e+01  0  0  0  0  0   0  0  0  0  0      0
MatPtAPNumeric    2 1.0 2.0182e+00 2.0 3.40e+08 1.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0 420671
VecDot           51 1.0 2.7399e-02 4.7 5.61e+02 1.2 0.0e+00 0.0e+00 5.1e+01  0  0  0  0  0   0  0  0  0  0     50
VecCopy          26 1.0 8.2970e-05 4.4 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
VecSet        40760 1.0 1.5796e-01 1.5 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
VecAXPY          52 1.0 3.2401e-04 6.9 6.48e+02 1.2 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0  4877
VecAYPX          29 1.0 1.3447e-04 3.5 3.36e+02 1.2 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0  6094
VecAssemblyBeg 5195 1.0 7.2004e+00 1.0 0.00e+00 0.0 1.3e+07 6.0e+00 7.8e+03  0  0  0  0 20   0  0  0  0 20      0
VecAssemblyEnd 5195 1.0 1.0332e+0012.1 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
VecScatterBeg 32965 1.0 3.7405e+02 5.2 0.00e+00 0.0 8.1e+10 1.6e+01 2.0e+04  5  0100100 52   5  0100100 52      0
VecScatterEnd 13013 1.0 4.3995e+03 1.1 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00 80  0  0  0  0  80  0  0  0  0      0
------------------------------------------------------------------------------------------------------------------------

Memory usage is given in bytes:

Object Type          Creations   Destructions     Memory  Descendants' Mem.
Reports information only for process 0.

--- Event Stage 0: Main Stage
```

```
              Matrix    49             49     480032400      0
              Vector  2685           2685      62282440      0
      Vector Scatter  2637           2637       1637020      0
           Index Set  2653           2653       2053092      0
              Viewer     1              0             0      0
========================================================================================================================
Average time to get PetscTime(): 1.90735e-07
Average time for MPI_Barrier(): 5.60284e-05
Average time for zero size MPI_Send(): 3.18241e-06
#PETSc Option Table entries:
-log_summary
#End of PETSc Option Table entries
Compiled without FORTRAN kernels
Compiled with full precision matrices (default)
sizeof(short) 2 sizeof(int) 4 sizeof(long) 8 sizeof(void*) 8 sizeof(PetscScalar) 8 sizeof(PetscInt) 4
Configure run at:
Configure options:
Application 4475766 resources: utime ~13040084s, stime ~55228s
------------------------------------------------------------------------------

Resources requested: mpparch=XT,mppnppn=32,mppwidth=2500,ncpus=1,place=pack,walltime=02:49:00
Resources allocated: cpupercent=0,cput=00:00:02,mem=5260kb,ncpus=1,vmem=39332kb,walltime=01:27:25

*** pr2c0008   Job: 1411448.sdb   ends: 30/04/13 14:33:49   queue: par:128n_3h ***
*** pr2c0008   Job: 1411448.sdb   ends: 30/04/13 14:33:49   queue: par:128n_3h ***
*** pr2c0008   Job: 1411448.sdb   ends: 30/04/13 14:33:49   queue: par:128n_3h ***
*** pr2c0008   Job: 1411448.sdb   ends: 30/04/13 14:33:49   queue: par:128n_3h ***
```

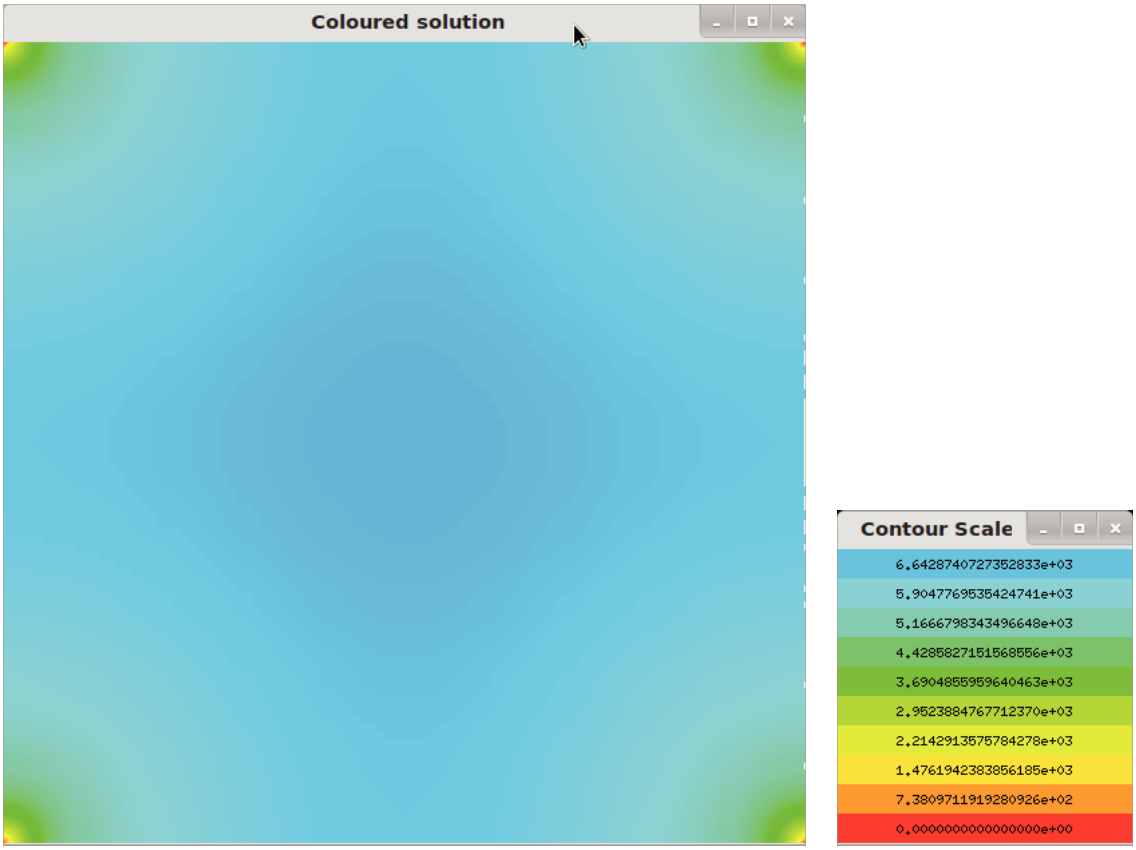


Figure A.1.: Main domain solved by benchmark.

A.2. Source code of my program

```
static char help[] = "FETI-DP method in 2D.\n(parallel processing)\n"; // works with Petsc-3.3-p6

#include <stdio.h>
#include <petscsnes.h>

int MatMatMultMPI();
void FEM();
int Add2Triangles();
int DirichletBoundaryCondition();
PetscInt BelongsToSubdomain();

#define NLVN 100 // number of local vertical nodes
#define NLVS 100 // number of local vertical subdomains
#define NLHN 100 // number of local horizontal nodes
#define NLHS 100 // number of local horizontal subdomains
#define Ns 10000  // number of subdomains

#define Nlambda 158400 // length of global vector lambda (number of remainder couples)
#define Nuc 10197      // length of global vector uc (number of corners)
#define Ndbc 4    // number of nodes with Dirichlet boundary conditions

PetscInt csc[Ns][4];      // global coordinates of subdomain left-top and right-bottom corner

PetscInt *gir[Nlambda];  //  global coordinates of remainder nodes couples
PetscInt *gic[Nuc]; //  global coordinates of corner nodes groups
                    // first number means how many nodes corner connects

PetscInt dbc[Ndbc][3]={{0,0,0},
                       {0,999,0},
                       {999,0,0},
                       {999,999,0}}; // global coordinates of nodes with Dirichlet boundary condition

// ------------------------------ main function --------------------------

PetscErrorCode  ierr;

int main(int argc,char **argv)
{    PetscInt i,j,k,l,m;  // just some integeres for various usage
   Mat tempMat1,tempMat2,tempMat3,tempMat4;  // just some matrices for various usage
   Mat I;               // some identity matrix
   Vec tempVec1,tempVec2,tempVec3,tempVec4; // just some vectors for various usage

   PetscInt NGVN=NLVN*NLVS;  // number of global vertical nodes
   PetscInt NGHN=NLHN*NLHS;  // number of global horizontal nodes

   PetscBool gsa[NGVN][NGHN]; // global shape array
   PetscBool *lsa; // pointers to local shape arrays
   PetscInt vSize; // vertical sizes of lsa
   PetscInt hSize; // horizontal sizes of lsa

   Mat Ks; // local stiffness matrices
   Vec fs; // local load vectors

   Mat Krr,Krc,Kcc;
   Mat Br,Bc;
   Vec fr,fbc,fc;
   Vec lambda,uc,ur,u;
   Vec uGlobal,uGlobalZero;

   Mat FIrc,Kccstar,Krrinv,Kccstarinv,Kccglobal;

   IS  is,is1,is2;       // needed for factorization
   MatFactorInfo info;   // needed for factorization

   Vec dr,fcstar;
   Mat FKF;
   Vec rhs;

   PetscDraw draw;
   PetscInt Nr;   // number of remainder nodes of subdomains
   PetscInt Nc;   // number of corner nodes of subdomains
   PetscInt Ne;   // number of empty nodes of subdomains
   PetscInt Ni;   // number of interior nodes of subdomains
   PetscInt N;    // number of all nodes of subdomains

   PetscInt *ir;   // indices of remainder nodes of subdomain
   PetscInt *ic;   // indices of corner nodes of subdomain
   PetscInt *ie;   // indices of empty nodes of subdomain
   PetscInt *ii;   // indices of interior nodes of subdomain

// ..........................................................................
   PetscInitialize(&argc,&argv,(char *)0,help);

   for(i=0;i<NGVN;i++){ for(j=0;j<NGHN;j++) gsa[i][j]=PETSC_TRUE; } // fill whole global shape array

   k=0;
```

```
  for(i=0;i<NGHN;i=i+NLHN){
    for(j=0;j<NGVN;j=j+NLVN){
      csc[k][0]=i;csc[k][1]=j;csc[k][2]=i+NLHN-1;csc[k][3]=j+NLVN-1; k++;
    }
  }

  k=0;
  for(i=NLHN;i<NGHN;i=i+NLHN){
    ierr = PetscMalloc(5*sizeof(PetscInt),&gic[k]);CHKERRQ(ierr);
    *gic[k]=2;*(gic[k]+1)=i-1;*(gic[k]+2)=0;*(gic[k]+3)=i;*(gic[k]+4)=0; k++;
    ierr = PetscMalloc(5*sizeof(PetscInt),&gic[k]);CHKERRQ(ierr);
    *gic[k]=2;*(gic[k]+1)=i-1;*(gic[k]+2)=NGHN-1;*(gic[k]+3)=i;*(gic[k]+4)=NGHN-1; k++;
  }

  for(i=NLVN;i<NGVN;i=i+NLVN){
    ierr = PetscMalloc(5*sizeof(PetscInt),&gic[k]);CHKERRQ(ierr);
    *gic[k]=2;*(gic[k]+1)=0;*(gic[k]+2)=i-1;*(gic[k]+3)=0;*(gic[k]+4)=i; k++;
    ierr = PetscMalloc(5*sizeof(PetscInt),&gic[k]);CHKERRQ(ierr);
    *gic[k]=2;*(gic[k]+1)=NGVN-1;*(gic[k]+2)=i-1;*(gic[k]+3)=NGVN-1;*(gic[k]+4)=i; k++;
  }

  for(i=NLHN;i<NGHN;i=i+NLHN){
    for(j=NLVN;j<NGVN;j=j+NLVN){
      ierr = PetscMalloc(9*sizeof(PetscInt),&gic[k]);CHKERRQ(ierr);
      *gic[k]=4;
      *(gic[k]+1)=i-1;*(gic[k]+2)=j-1;
      *(gic[k]+3)=i;*(gic[k]+4)=j-1;
      *(gic[k]+5)=i-1;*(gic[k]+6)=j;
      *(gic[k]+7)=i;*(gic[k]+8)=j;
      k++;
    }
  }

  PetscBool isCorner;
  m=0;
  for(i=0;i<NGHN;i++){
    for(j=NLVN;j<NGVN;j=j+NLVN){
      isCorner=PETSC_FALSE;
      for(k=0;k<Nuc;k++){
        for(l=0;l<*gic[k];l++){
          if(((j-1)==*(gic[k]+2*l+1))&&(i==*(gic[k]+2*l+2))) {isCorner=PETSC_TRUE;}
          if((j==*(gic[k]+2*l+1))&&(i==*(gic[k]+2*l+2))) {isCorner=PETSC_TRUE;}
        }
      }
      if(!(isCorner)) {
        ierr = PetscMalloc(4*sizeof(PetscInt),&gir[m]);CHKERRQ(ierr);
        *gir[m]=j-1;*(gir[m]+1)=i;*(gir[m]+2)=j;*(gir[m]+3)=i; m++;
      }
    }
  }

  for(i=0;i<NGVN;i++){
    for(j=NLHN;j<NGHN;j=j+NLHN){
      isCorner=PETSC_FALSE;
      for(k=0;k<Nuc;k++){
        for(l=0;l<*gic[k];l++){
          if((i==*(gic[k]+2*l+1))&&((j-1)==*(gic[k]+2*l+2))) {isCorner=PETSC_TRUE;}
          if((i==*(gic[k]+2*l+1))&&(j==*(gic[k]+2*l+2))) {isCorner=PETSC_TRUE;}
        }
      }
      if(!(isCorner)) {
        ierr = PetscMalloc(4*sizeof(PetscInt),&gir[m]);CHKERRQ(ierr);
        *gir[m]=i;*(gir[m]+1)=j-1;*(gir[m]+2)=i;*(gir[m]+3)=j; m++;
      }
    }
  }

  PetscMPIInt r;
  MPI_Comm_rank(PETSC_COMM_WORLD,&r);

  Nr=0;Nc=0;Ne=0;
  vSize=csc[r][2]-csc[r][0]+1;
  hSize=csc[r][3]-csc[r][1]+1;
       N=vSize*hSize;
  ierr = PetscMalloc(N*sizeof(PetscBool),&lsa);CHKERRQ(ierr); // create empty local shape array
  for(j=0;j<vSize;j++){
    for(k=0;k<hSize;k++){
       *(lsa+j*hSize+k)=gsa[j+csc[r][0]][k+csc[r][1]];  // fill local shape array from global shape array
       if(!(*(lsa+j*hSize+k))){Ne++;}
    }
  }

// ..........................................................................

  for(i=0;i<Nlambda;i++){
    if(r==BelongsToSubdomain(*gir[i],*(gir[i]+1))) Nr++;
    if(r==BelongsToSubdomain(*(gir[i]+2),*(gir[i]+3))) Nr++;
  }

  for(i=0;i<Nuc;i++){
```

```
      for(j=0;j<*gic[i];j++){
        if(r==BelongsToSubdomain(*(gic[i]+2*j+1),*(gic[i]+2*j+2))) Nc++;
      }
    }

    Ni=N-Nr-Nc-Ne;

// ...........................................................................

    ierr = PetscMalloc(Nr*sizeof(PetscInt),&ir);CHKERRQ(ierr);   // allocating memory for arrays of indices
    ierr = PetscMalloc(Nc*sizeof(PetscInt),&ic);CHKERRQ(ierr);
    ierr = PetscMalloc(Ni*sizeof(PetscInt),&ii);CHKERRQ(ierr);
    ierr = PetscMalloc(Ne*sizeof(PetscInt),&ie);CHKERRQ(ierr);

// ...........................................................................
// computing remainder nodes indices ir[]

    i=0;
    for(j=0;j<Nlambda;j++){
      if(r==BelongsToSubdomain(*gir[j],*(gir[j]+1))){
        *(ir+i)=(*gir[j]-csc[r][0])*hSize+*(gir[j]+1)-csc[r][1];
        i++;
      }
      if(r==BelongsToSubdomain(*(gir[j]+2),*(gir[j]+3))){
        *(ir+i)=(*(gir[j]+2)-csc[r][0])*hSize+*(gir[j]+3)-csc[r][1];
        i++;
      }
    }

// computing corner nodes indices ic[]

    i=0;
    for(j=0;j<Nuc;j++){
      for(l=0;l<*gic[j];l++){
        if(r==BelongsToSubdomain(*(gic[j]+2*l+1),*(gic[j]+2*l+2))){
          *(ic+i)=(*(gic[j]+2*l+1)-csc[r][0])*hSize+*(gic[j]+2*l+2)-csc[r][1];
          i++;
        }
      }
    }

    i=0;
    for(j=0;j<N;j++){
      if(!(*(lsa+j))){*(ie+i)=j;i++;}
    }

    PetscInt *t;
    ierr = PetscMalloc(N*sizeof(PetscInt),&t);CHKERRQ(ierr);  // creating temporary array of all indices of subdomain
    for(i=0;i<N;i++){ *(t+i)=i;}
    for(i=0;i<Nr;i++){ *(t+*(ir+i))=-1;}
    for(i=0;i<Nc;i++){ *(t+*(ic+i))=-1;}
    for(i=0;i<Ne;i++){ *(t+*(ie+i))=-1;}
    ierr = PetscFree(ie);CHKERRQ(ierr);
    for(i=0;i<N;i++){
      if(*(t+i)==-1){                              // releasing remainder, corner and empty nodes indices
        for(k=i;k<N;k++){
          *(t+k)=*(t+k+1);
        }        i--;
      }
    }
    ierr = PetscMemcpy(ii,t,Ni*sizeof(PetscInt));CHKERRQ(ierr);   // saving interior indices
    ierr = PetscFree(t);CHKERRQ(ierr);

// ...........................................................................................
// ...........................................................................................

    PetscPrintf(PETSC_COMM_WORLD,"\nComputing local stiffness matrices Ks: ");
    ierr = MatCreate(PETSC_COMM_SELF,&Ks);CHKERRQ(ierr); // create empty local stiffness matrix Ks
    ierr = MatSetSizes(Ks,PETSC_DECIDE,PETSC_DECIDE,N,N);CHKERRQ(ierr);
    ierr = MatSetType(Ks,MATSEQDENSE);CHKERRQ(ierr);
    ierr = MatSetUp(Ks);CHKERRQ(ierr);
    ierr = MatZeroEntries(Ks);CHKERRQ(ierr);
    ierr = VecCreate(PETSC_COMM_SELF,&fs);CHKERRQ(ierr); // create empty local load vector fs
    ierr = VecSetSizes(fs,PETSC_DECIDE,N);CHKERRQ(ierr);
    ierr = VecSetType(fs,VECSEQ);CHKERRQ(ierr);
    ierr = VecZeroEntries(fs);CHKERRQ(ierr);
    FEM(Ks,fs,lsa,vSize,hSize);                    // fill Ks and fs by finite element method
    ierr = PetscFree(lsa);CHKERRQ(ierr);
    PetscPrintf(PETSC_COMM_WORLD," done.");

// ...........................................................................
// adjusting Ks and fs to satisfy Dirichlet boundary condition on certain nodes

    for(i=0;i<Ndbc;i++){
      if(r==BelongsToSubdomain(dbc[i][0],dbc[i][1])){
        DirichletBoundaryCondition(Ks,fs,(dbc[i][0]-csc[r][0])*hSize+dbc[i][1]-csc[r][1],dbc[i][2]);
      }
    }

// ...........................................................................
```

```
// filling Krr,Krc,Kcc,fr,fbc of each subdomain

   ierr = PetscMalloc((Ni+Nr)*sizeof(PetscInt),&t);CHKERRQ(ierr);
   for(i=0;i<Ni;i++){*(t+i)=*(ii+i);}     // adding interior nodes indices
   for(i=0;i<Nr;i++){*(t+Ni+i)=*(ir+i);}  // adding remainder nodes indices

   IS tempISir,tempISc;
   ierr = ISCreateGeneral(PETSC_COMM_SELF,Ni+Nr,t,PETSC_COPY_VALUES,&tempISir);CHKERRQ(ierr);
   ierr = ISCreateGeneral(PETSC_COMM_SELF,Nc,ic,PETSC_COPY_VALUES,&tempISc);CHKERRQ(ierr);
   ierr = PetscFree(t);CHKERRQ(ierr);

   ierr = MatAssemblyBegin(Ks,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatAssemblyEnd(Ks,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatGetSubMatrix(Ks,tempISir,tempISir,MAT_INITIAL_MATRIX,&Krr);CHKERRQ(ierr);
   ierr = MatGetSubMatrix(Ks,tempISir,tempISc,MAT_INITIAL_MATRIX,&Krc);CHKERRQ(ierr);
   ierr = MatGetSubMatrix(Ks,tempISc,tempISc,MAT_INITIAL_MATRIX,&Kcc);CHKERRQ(ierr);
   ierr = MatDestroy(&Ks);CHKERRQ(ierr);
   ierr = VecGetSubVector(fs,tempISir,&fr);CHKERRQ(ierr);
   ierr = VecGetSubVector(fs,tempISc,&fbc);CHKERRQ(ierr);
   ierr = VecDestroy(&fs);CHKERRQ(ierr);
   ierr = ISDestroy(&tempISir);CHKERRQ(ierr);
   ierr = ISDestroy(&tempISc);CHKERRQ(ierr);

// ..........................................................................

   PetscPrintf(PETSC_COMM_WORLD,"\nInverting local matrices Krr: ");
   ierr = MatCreate(PETSC_COMM_SELF,&I);CHKERRQ(ierr);
   ierr = MatSetType(I,MATSEQDENSE);CHKERRQ(ierr);
   ierr = MatSetSizes(I,PETSC_DECIDE,PETSC_DECIDE,Ni+Nr,Ni+Nr);CHKERRQ(ierr);
   ierr = MatSetUp(I);CHKERRQ(ierr);
   ierr = MatZeroEntries(I);CHKERRQ(ierr);
   ierr = MatShift(I,1.0);CHKERRQ(ierr);                    // creating identity matrix
   ierr = MatAssemblyBegin(I,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatAssemblyEnd(I,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatCreate(PETSC_COMM_SELF,&Krrinv);CHKERRQ(ierr);
   ierr = MatSetSizes(Krrinv,PETSC_DECIDE,PETSC_DECIDE,Ni+Nr,Ni+Nr);CHKERRQ(ierr);
   ierr = MatSetType(Krrinv,MATSEQDENSE);CHKERRQ(ierr);
   ierr = MatSetUp(Krrinv);CHKERRQ(ierr);
   ierr = MatGetOrdering(Krr,MATORDERINGNATURAL,&is1,&is2);CHKERRQ(ierr);
   ierr = MatFactorInfoInitialize(&info);CHKERRQ(ierr);
   ierr = MatLUFactor(Krr,is1,is2,&info);CHKERRQ(ierr);
   ierr = ISDestroy(&is1);CHKERRQ(ierr);
   ierr = ISDestroy(&is2);CHKERRQ(ierr);
   ierr = MatMatSolve(Krr,I,Krrinv);CHKERRQ(ierr);       // inverting Krr[] --> Krrinv[]
   ierr = MatDestroy(&I);CHKERRQ(ierr);
   ierr = MatDestroy(&Krr);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD," done.");

// ..........................................................................
// creating local matrices Br,Bc for each subdomain

   ierr = MatCreate(PETSC_COMM_SELF,&Br);CHKERRQ(ierr);
   ierr = MatSetSizes(Br,PETSC_DECIDE,PETSC_DECIDE,Nlambda,Ni+Nr);CHKERRQ(ierr);
   ierr = MatSetType(Br,MATSEQDENSE);CHKERRQ(ierr);
   ierr = MatSetUp(Br);CHKERRQ(ierr);
   ierr = MatZeroEntries(Br);CHKERRQ(ierr);
   ierr = MatCreate(PETSC_COMM_SELF,&Bc);CHKERRQ(ierr);
   ierr = MatSetSizes(Bc,PETSC_DECIDE,PETSC_DECIDE,Nc,Nuc);CHKERRQ(ierr);
   ierr = MatSetType(Bc,MATSEQDENSE);CHKERRQ(ierr);
   ierr = MatSetUp(Bc);CHKERRQ(ierr);
   ierr = MatZeroEntries(Bc);CHKERRQ(ierr);

   PetscPrintf(PETSC_COMM_WORLD,"\nCreating matrices Br: ");
   for(i=0;i<Nlambda;i++){                  // adjusting Br[]
     if(r==BelongsToSubdomain(*gir[i],*(gir[i]+1))){
       k=(*gir[i]-csc[r][0])*hSize+*(gir[i]+1)-csc[r][1];
       for(l=0;l<Nr;l++){
         if(*(ir+l)==k){MatSetValue(Br,i,Ni+l,1.0,INSERT_VALUES);}
       }
     }
     if(r==BelongsToSubdomain(*(gir[i]+2),*(gir[i]+3))){
       k=(*(gir[i]+2)-csc[r][0])*hSize+*(gir[i]+3)-csc[r][1];
       for(l=0;l<Nr;l++){
         if(*(ir+l)==k){MatSetValue(Br,i,Ni+l,-1.0,INSERT_VALUES);}
       }
     }
   }
   for(i=0;i<Nlambda;i++) PetscFree(gir[i]);
   PetscPrintf(PETSC_COMM_WORLD," done.");

// ..........................................................................

   PetscPrintf(PETSC_COMM_WORLD,"\nCreating matrices Bc: ");
   for(i=0;i<Nuc;i++){                      // adjusting Bc[]
     for(k=0;k<*gic[i];k++){
       if(r==BelongsToSubdomain(*(gic[i]+2*k+1),*(gic[i]+2*k+2))){
       l=(*(gic[i]+2*k+1)-csc[r][0])*hSize+*(gic[i]+2*k+2)-csc[r][1];
         for(m=0;m<Nc;m++){
           if(*(ic+m)==l){MatSetValue(Bc,m,i,1.0,INSERT_VALUES);}
         }
```

```
      }
    }
  }
  for(i=0;i<Nuc;i++) PetscFree(gic[i]);
  PetscPrintf(PETSC_COMM_WORLD," done.");

// ..........................................................................

  VecScatter vecscatNuc;
  VecScatter vecscatNlambda;
  VecScatter vecscatToAll;
  VecScatter vecscatToZero;

  ierr = VecCreate(PETSC_COMM_SELF,&tempVec1);CHKERRQ(ierr);
  ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
  ierr = VecSetType(tempVec1,VECSEQ);CHKERRQ(ierr);
  ierr = VecCreate(PETSC_COMM_WORLD,&fc);CHKERRQ(ierr);
  ierr = VecSetSizes(fc,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
  ierr = VecSetType(fc,VECMPI);CHKERRQ(ierr);
  ierr = VecZeroEntries(fc);CHKERRQ(ierr);
  PetscPrintf(PETSC_COMM_WORLD,"\nComputing global vector fc: ");

  ierr = MatAssemblyBegin(Bc,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
  ierr = MatAssemblyEnd(Bc,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
  ierr = MatMultTranspose(Bc,fbc,tempVec1);CHKERRQ(ierr);   // adjusting fc
  ierr = VecDestroy(&fbc);CHKERRQ(ierr);

  PetscInt ArrNuc[Nuc];
  for(i=0;i<Nuc;i++){ArrNuc[i]=i;}
  ierr = ISCreateGeneral(PETSC_COMM_WORLD,Nuc,ArrNuc,PETSC_COPY_VALUES,&is);CHKERRQ(ierr);
  ierr = VecScatterCreate(tempVec1,PETSC_NULL,fc,is,&vecscatNuc);CHKERRQ(ierr);
  ierr = ISDestroy(&is);CHKERRQ(ierr);
  ierr = VecScatterBegin(vecscatNuc,tempVec1,fc,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
  ierr = VecScatterEnd(vecscatNuc,tempVec1,fc,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
  PetscPrintf(PETSC_COMM_WORLD," done.");
  VecDestroy(&tempVec1);

// ..........................................................................
// global matrix Kccglobal

  ierr = MatCreate(PETSC_COMM_WORLD,&Kccglobal);CHKERRQ(ierr);
  ierr = MatSetSizes(Kccglobal,PETSC_DECIDE,PETSC_DECIDE,Nuc,Nuc);CHKERRQ(ierr);
  ierr = MatSetType(Kccglobal,MATMPIAIJ);CHKERRQ(ierr);
  ierr = MatSetUp(Kccglobal);CHKERRQ(ierr);
  PetscPrintf(PETSC_COMM_WORLD,"\nComputing global matrix Kcc: ");
  ierr = MatConvert(Kcc,MATSEQAIJ,MAT_INITIAL_MATRIX,&tempMat1);CHKERRQ(ierr);
  ierr = MatDestroy(&Kcc);CHKERRQ(ierr);
  ierr = MatConvert(Bc,MATSEQAIJ,MAT_INITIAL_MATRIX,&tempMat2);CHKERRQ(ierr);
  ierr = MatPtAP(tempMat1,tempMat2,MAT_INITIAL_MATRIX,1.0,&tempMat3);CHKERRQ(ierr);
  ierr = MatDestroy(&tempMat1);CHKERRQ(ierr);
  ierr = MatDestroy(&tempMat2);CHKERRQ(ierr);

  PetscScalar *tempArr;
  PetscInt *tempArrIdx;
  PetscInt istart,iend;
  ierr = VecCreate(PETSC_COMM_WORLD,&tempVec1);CHKERRQ(ierr);
  ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
  ierr = VecSetType(tempVec1,VECMPI);CHKERRQ(ierr);
  ierr = VecCreate(PETSC_COMM_SELF,&tempVec2);CHKERRQ(ierr);
  ierr = VecSetSizes(tempVec2,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
  ierr = VecSetType(tempVec2,VECSEQ);CHKERRQ(ierr);
  ierr = VecCreate(PETSC_COMM_SELF,&tempVec3);CHKERRQ(ierr);
  ierr = VecSetSizes(tempVec3,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
  ierr = VecSetType(tempVec3,VECSEQ);CHKERRQ(ierr);

  ierr = PetscMalloc(Nuc*sizeof(PetscScalar),&tempArr);CHKERRQ(ierr);
  ierr = PetscMalloc(Nuc*sizeof(PetscInt),&tempArrIdx);CHKERRQ(ierr);
  for(i=0;i<Nuc;i++){
    ierr = VecZeroEntries(tempVec1);CHKERRQ(ierr);
    ierr = VecZeroEntries(tempVec2);CHKERRQ(ierr);
    ierr = VecSetValue(tempVec2,i,1.0,INSERT_VALUES);CHKERRQ(ierr);
    ierr = MatMult(tempMat3,tempVec2,tempVec3);CHKERRQ(ierr);
    ierr = VecScatterBegin(vecscatNuc,tempVec3,tempVec1,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
    ierr = VecScatterEnd(vecscatNuc,tempVec3,tempVec1,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
    ierr = VecGetOwnershipRange(tempVec1,&istart,&iend);CHKERRQ(ierr);
    ierr = VecGetArray(tempVec1,&tempArr);CHKERRQ(ierr);
    for(j=istart;j<iend;j++){tempArrIdx[j-istart]=j;}
    ierr = MatSetValues(Kccglobal,iend-istart,tempArrIdx,1,&i,tempArr,INSERT_VALUES);CHKERRQ(ierr);  // adjusting Kccglobal
    ierr = VecRestoreArray(tempVec1,&tempArr);CHKERRQ(ierr);
  }
  ierr = VecDestroy(&tempVec3);CHKERRQ(ierr);
  ierr = VecDestroy(&tempVec2);CHKERRQ(ierr);
  ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
  ierr = MatDestroy(&tempMat3);CHKERRQ(ierr);
  ierr = PetscFree(tempArr);CHKERRQ(ierr);
  ierr = PetscFree(tempArrIdx);CHKERRQ(ierr);
  PetscPrintf(PETSC_COMM_WORLD," done.");
  ierr = MatAssemblyBegin(Kccglobal,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
  ierr = MatAssemblyEnd(Kccglobal,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
```

```
// ..........................................................................
// global matrix FIrc

   ierr = MatCreate(PETSC_COMM_WORLD,&FIrc);CHKERRQ(ierr);
   ierr = MatSetSizes(FIrc,PETSC_DECIDE,PETSC_DECIDE,Nlambda,Nuc);CHKERRQ(ierr);
   ierr = MatSetType(FIrc,MATMPIDENSE);CHKERRQ(ierr);
   ierr = MatSetUp(FIrc);CHKERRQ(ierr);

   PetscPrintf(PETSC_COMM_WORLD,"\nComputing global matrix FIrc: ");
   ierr = MatAssemblyBegin(Br,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatAssemblyEnd(Br,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatMatMult(Br,Krrinv,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&tempMat1);CHKERRQ(ierr);
   ierr = MatMatMult(tempMat1,Krc,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&tempMat2);CHKERRQ(ierr);
   ierr = MatMatMult(tempMat2,Bc,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&tempMat3);CHKERRQ(ierr);

   ierr = VecCreate(PETSC_COMM_WORLD,&tempVec1);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nlambda);CHKERRQ(ierr);
   ierr = VecSetType(tempVec1,VECMPI);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec3);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec3,PETSC_DECIDE,Nlambda);CHKERRQ(ierr);
   ierr = VecSetType(tempVec3,VECSEQ);CHKERRQ(ierr);

   PetscInt ArrNlambda[Nlambda];
   for(i=0;i<Nlambda;i++){ArrNlambda[i]=i;}
   ierr = ISCreateGeneral(PETSC_COMM_WORLD,Nlambda,ArrNlambda,PETSC_COPY_VALUES,&is);CHKERRQ(ierr);
   ierr = VecScatterCreate(tempVec3,PETSC_NULL,tempVec1,is,&vecscatNlambda);CHKERRQ(ierr);
   ierr = ISDestroy(&is);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec2);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec2,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
   ierr = VecSetType(tempVec2,VECSEQ);CHKERRQ(ierr);
   ierr = PetscMalloc(Nlambda*sizeof(PetscScalar),&tempArr);CHKERRQ(ierr);
   ierr = PetscMalloc(Nlambda*sizeof(PetscInt),&tempArrIdx);CHKERRQ(ierr);
   for(i=0;i<Nuc;i++){
     ierr = VecZeroEntries(tempVec1);CHKERRQ(ierr);
     ierr = VecZeroEntries(tempVec2);CHKERRQ(ierr);
     ierr = VecSetValue(tempVec2,i,1.0,INSERT_VALUES);CHKERRQ(ierr);
     ierr = MatMult(tempMat3,tempVec2,tempVec3);CHKERRQ(ierr);
     ierr = VecScatterBegin(vecscatNlambda,tempVec3,tempVec1,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
     ierr = VecScatterEnd(vecscatNlambda,tempVec3,tempVec1,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
     ierr = VecGetOwnershipRange(tempVec1,&istart,&iend);CHKERRQ(ierr);
     ierr = VecGetArray(tempVec1,&tempArr);CHKERRQ(ierr);
     for(j=istart;j<iend;j++){tempArrIdx[j-istart]=j;}
     ierr = MatSetValues(FIrc,iend-istart,tempArrIdx,1,&i,tempArr,INSERT_VALUES);CHKERRQ(ierr); // adjusting FIrc
     ierr = VecRestoreArray(tempVec1,&tempArr);CHKERRQ(ierr);
   }
   ierr = VecDestroy(&tempVec3);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec2);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
   ierr = MatDestroy(&tempMat3);CHKERRQ(ierr);
   ierr = MatDestroy(&tempMat2);CHKERRQ(ierr);
   ierr = MatDestroy(&tempMat1);CHKERRQ(ierr);
   ierr = PetscFree(tempArr);CHKERRQ(ierr);
   ierr = PetscFree(tempArrIdx);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD," done.");
   ierr = MatAssemblyBegin(FIrc,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatAssemblyEnd(FIrc,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);

// ..........................................................................
// global matrix Kccstar

   ierr = MatCreate(PETSC_COMM_WORLD,&Kccstar);CHKERRQ(ierr);
   ierr = MatSetSizes(Kccstar,PETSC_DECIDE,PETSC_DECIDE,Nuc,Nuc);CHKERRQ(ierr);
   ierr = MatSetType(Kccstar,MATMPIAIJ);CHKERRQ(ierr);
   ierr = MatSetUp(Kccstar);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD,"\nComputing global matrix Kcc*: ");
   ierr = MatMatMult(Krc,Bc,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&tempMat1);CHKERRQ(ierr);
   ierr = MatConvert(Krrinv,MATSEQAIJ,MAT_INITIAL_MATRIX,&tempMat2);CHKERRQ(ierr);
   ierr = MatConvert(tempMat1,MATSEQAIJ,MAT_INITIAL_MATRIX,&tempMat3);CHKERRQ(ierr);
   ierr = MatPtAP(tempMat2,tempMat3,MAT_INITIAL_MATRIX,1.0,&tempMat4);CHKERRQ(ierr);

   ierr = VecCreate(PETSC_COMM_WORLD,&tempVec1);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
   ierr = VecSetType(tempVec1,VECMPI);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec2);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec2,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
   ierr = VecSetType(tempVec2,VECSEQ);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec3);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec3,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
   ierr = VecSetType(tempVec3,VECSEQ);CHKERRQ(ierr);

   ierr = PetscMalloc(Nuc*sizeof(PetscScalar),&tempArr);CHKERRQ(ierr);
   ierr = PetscMalloc(Nuc*sizeof(PetscInt),&tempArrIdx);CHKERRQ(ierr);
   for(i=0;i<Nuc;i++){
     ierr = VecZeroEntries(tempVec1);CHKERRQ(ierr);
     ierr = VecZeroEntries(tempVec2);CHKERRQ(ierr);
     ierr = VecSetValue(tempVec2,i,1.0,INSERT_VALUES);CHKERRQ(ierr);
     ierr = MatMult(tempMat4,tempVec2,tempVec3);CHKERRQ(ierr);
     ierr = VecScatterBegin(vecscatNuc,tempVec3,tempVec1,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
     ierr = VecScatterEnd(vecscatNuc,tempVec3,tempVec1,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
     ierr = VecGetOwnershipRange(tempVec1,&istart,&iend);CHKERRQ(ierr);
```

```
      ierr = VecGetArray(tempVec1,&tempArr);CHKERRQ(ierr);
      for(j=istart;j<iend;j++){tempArrIdx[j-istart]=j;}
      ierr = MatSetValues(Kccstar,iend-istart,tempArrIdx,1,&i,tempArr,INSERT_VALUES);CHKERRQ(ierr); // adjusting Kcc*
      ierr = VecRestoreArray(tempVec1,&tempArr);CHKERRQ(ierr);
    }
    ierr = PetscFree(tempArr);CHKERRQ(ierr);
    ierr = PetscFree(tempArrIdx);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec3);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec2);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
    ierr = MatDestroy(&tempMat4);CHKERRQ(ierr);
    ierr = MatDestroy(&tempMat3);CHKERRQ(ierr);
    ierr = MatDestroy(&tempMat2);CHKERRQ(ierr);
    ierr = MatDestroy(&tempMat1);CHKERRQ(ierr);

    ierr = MatAssemblyBegin(Kccstar,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
    ierr = MatAssemblyEnd(Kccstar,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
    ierr = MatAYPX(Kccstar,-1.0,Kccglobal,DIFFERENT_NONZERO_PATTERN);CHKERRQ(ierr);  // final adjustment of Kcstar
    ierr = MatDestroy(&Kccglobal);CHKERRQ(ierr);
    PetscPrintf(PETSC_COMM_WORLD," done.");

// ..........................................................................
// global vector dr

    ierr = VecCreate(PETSC_COMM_SELF,&tempVec1);CHKERRQ(ierr);
    ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nlambda);CHKERRQ(ierr);
    ierr = VecSetType(tempVec1,VECSEQ);CHKERRQ(ierr);
    ierr = VecCreate(PETSC_COMM_WORLD,&dr);CHKERRQ(ierr);
    ierr = VecSetSizes(dr,PETSC_DECIDE,Nlambda);CHKERRQ(ierr);
    ierr = VecSetType(dr,VECMPI);CHKERRQ(ierr);

    PetscPrintf(PETSC_COMM_WORLD,"\nComputing global vector dr: ");
    ierr = MatMatMult(Br,Krrinv,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&tempMat1);CHKERRQ(ierr);
    ierr = MatMult(tempMat1,fr,tempVec1);CHKERRQ(ierr);
    ierr = VecScatterBegin(vecscatNlambda,tempVec1,dr,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);   // adjusting dr
    ierr = VecScatterEnd(vecscatNlambda,tempVec1,dr,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
    ierr = MatDestroy(&tempMat1);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
    PetscPrintf(PETSC_COMM_WORLD," done.");

// ..........................................................................
// global vector fcstar

    ierr = VecCreate(PETSC_COMM_WORLD,&fcstar);CHKERRQ(ierr);
    ierr = VecSetSizes(fcstar,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
    ierr = VecSetType(fcstar,VECMPI);CHKERRQ(ierr);
    ierr = VecCreate(PETSC_COMM_SELF,&tempVec1);CHKERRQ(ierr);
    ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
    ierr = VecSetType(tempVec1,VECSEQ);CHKERRQ(ierr);
    PetscPrintf(PETSC_COMM_WORLD,"\nComputing global vector fc*: ");
    ierr = MatMatMult(Krc,Bc,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&tempMat1);CHKERRQ(ierr);
    ierr = MatTranspose(tempMat1,MAT_INITIAL_MATRIX,&tempMat2);CHKERRQ(ierr);
    ierr = MatMatMult(tempMat2,Krrinv,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&tempMat3);CHKERRQ(ierr);
    ierr = MatMult(tempMat3,fr,tempVec1);CHKERRQ(ierr);   // adjusting fcstar
    ierr = MatDestroy(&tempMat3);CHKERRQ(ierr);
    ierr = MatDestroy(&tempMat2);CHKERRQ(ierr);
    ierr = MatDestroy(&tempMat1);CHKERRQ(ierr);
    ierr = VecScatterBegin(vecscatNuc,tempVec1,fcstar,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
    ierr = VecScatterEnd(vecscatNuc,tempVec1,fcstar,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
    ierr = VecAYPX(fcstar,-1.0,fc);CHKERRQ(ierr);     // final adjustment of fcstar
    ierr = VecDestroy(&fc);CHKERRQ(ierr);
    PetscPrintf(PETSC_COMM_WORLD," done.");

// ..........................................................................

    PetscPrintf(PETSC_COMM_WORLD,"\nInverting global matrix Kcc*: ");
    ierr = VecCreate(PETSC_COMM_WORLD,&tempVec1);CHKERRQ(ierr);
    ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
    ierr = VecSetType(tempVec1,VECMPI);CHKERRQ(ierr);
    ierr = VecCreate(PETSC_COMM_WORLD,&tempVec2);CHKERRQ(ierr);
    ierr = VecSetSizes(tempVec2,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
    ierr = VecSetType(tempVec2,VECMPI);CHKERRQ(ierr);
    ierr = MatCreate(PETSC_COMM_SELF,&tempMat1);CHKERRQ(ierr);
    ierr = MatSetType(tempMat1,MATSEQDENSE);CHKERRQ(ierr);
    ierr = MatSetSizes(tempMat1,PETSC_DECIDE,PETSC_DECIDE,Nuc,Nuc);CHKERRQ(ierr);
    ierr = MatSetUp(tempMat1);CHKERRQ(ierr);
    ierr = PetscMalloc(Nuc*sizeof(PetscScalar),&tempArr);CHKERRQ(ierr);
    for(i=0;i<Nuc;i++){
      ierr = VecZeroEntries(tempVec1);CHKERRQ(ierr);
      ierr = VecSetValue(tempVec1,i,1.0,INSERT_VALUES);CHKERRQ(ierr);
      ierr = VecAssemblyBegin(tempVec1);CHKERRQ(ierr);
      ierr = VecAssemblyEnd(tempVec1);CHKERRQ(ierr);
      ierr = MatMult(Kccstar,tempVec1,tempVec2);CHKERRQ(ierr);
      ierr = VecScatterCreateToAll(tempVec2,&vecscatToAll,&tempVec3);CHKERRQ(ierr);
      ierr = VecScatterBegin(vecscatToAll,tempVec2,tempVec3,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
      ierr = VecScatterEnd(vecscatToAll,tempVec2,tempVec3,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
      ierr = VecGetArray(tempVec3,&tempArr);CHKERRQ(ierr);
      ierr = MatSetValues(tempMat1,Nuc,ArrNuc,1,&i,tempArr,INSERT_VALUES);CHKERRQ(ierr);
      ierr = VecRestoreArray(tempVec3,&tempArr);CHKERRQ(ierr);
```

```
      ierr = VecDestroy(&tempVec3);CHKERRQ(ierr);
      ierr = VecScatterDestroy(&vecscatToAll);CHKERRQ(ierr);
    }
    ierr = MatDestroy(&Kccstar);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec2);CHKERRQ(ierr);
    ierr = MatAssemblyBegin(tempMat1,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
    ierr = MatAssemblyEnd(tempMat1,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);

    ierr = MatCreate(PETSC_COMM_SELF,&I);CHKERRQ(ierr);
    ierr = MatSetType(I,MATSEQDENSE);CHKERRQ(ierr);
    ierr = MatSetSizes(I,PETSC_DECIDE,PETSC_DECIDE,Nuc,Nuc);CHKERRQ(ierr);
    ierr = MatSetUp(I);CHKERRQ(ierr);
    ierr = MatZeroEntries(I);CHKERRQ(ierr);
    ierr = MatAssemblyBegin(I,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
    ierr = MatAssemblyEnd(I,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
    ierr = MatShift(I,1.0);CHKERRQ(ierr);                          // creating identity matrix
    ierr = MatCreate(PETSC_COMM_SELF,&tempMat2);CHKERRQ(ierr);
    ierr = MatSetSizes(tempMat2,PETSC_DECIDE,PETSC_DECIDE,Nuc,Nuc);CHKERRQ(ierr);
    ierr = MatSetType(tempMat2,MATSEQDENSE);CHKERRQ(ierr);
    ierr = MatSetUp(tempMat2);CHKERRQ(ierr);
    ierr = MatGetOrdering(tempMat1,MATORDERINGNATURAL,&is1,&is2);CHKERRQ(ierr);
    ierr = MatFactorInfoInitialize(&info);CHKERRQ(ierr);
    ierr = MatLUFactor(tempMat1,is1,is2,&info);CHKERRQ(ierr);
    ierr = ISDestroy(&is1);CHKERRQ(ierr);
    ierr = ISDestroy(&is2);CHKERRQ(ierr);
    ierr = MatMatSolve(tempMat1,I,tempMat2);CHKERRQ(ierr);    // inverting Kcstar --> Kcstarinv
    ierr = MatDestroy(&tempMat1);CHKERRQ(ierr);
    ierr = MatDestroy(&I);CHKERRQ(ierr);

    ierr = VecCreate(PETSC_COMM_SELF,&tempVec1);CHKERRQ(ierr);
    ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
    ierr = VecSetType(tempVec1,VECSEQ);CHKERRQ(ierr);
    ierr = VecCreate(PETSC_COMM_SELF,&tempVec2);CHKERRQ(ierr);
    ierr = VecSetSizes(tempVec2,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
    ierr = VecSetType(tempVec2,VECSEQ);CHKERRQ(ierr);
    ierr = VecCreate(PETSC_COMM_WORLD,&tempVec3);CHKERRQ(ierr);
    ierr = VecSetSizes(tempVec3,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
    ierr = VecSetType(tempVec3,VECMPI);CHKERRQ(ierr);
    ierr = MatCreate(PETSC_COMM_WORLD,&Kccstarinv);CHKERRQ(ierr);
    ierr = MatSetSizes(Kccstarinv,PETSC_DECIDE,PETSC_DECIDE,Nuc,Nuc);CHKERRQ(ierr);
    ierr = MatSetType(Kccstarinv,MATMPIDENSE);CHKERRQ(ierr);
    ierr = MatSetUp(Kccstarinv);CHKERRQ(ierr);
    ierr = PetscMalloc(Nuc*sizeof(PetscInt),&tempArrIdx);CHKERRQ(ierr);
    for(i=0;i<Nuc;i++){
      ierr = VecZeroEntries(tempVec1);CHKERRQ(ierr);
      ierr = VecSetValue(tempVec1,i,1.0,INSERT_VALUES);CHKERRQ(ierr);
      ierr = VecAssemblyBegin(tempVec1);CHKERRQ(ierr);
      ierr = VecAssemblyEnd(tempVec1);CHKERRQ(ierr);
      ierr = MatMult(tempMat2,tempVec1,tempVec2);CHKERRQ(ierr);
      ierr = VecScatterBegin(vecscatNuc,tempVec2,tempVec3,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
      ierr = VecScatterEnd(vecscatNuc,tempVec2,tempVec3,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
      ierr = VecGetOwnershipRange(tempVec3,&istart,&iend);CHKERRQ(ierr);
      ierr = VecGetArray(tempVec3,&tempArr);CHKERRQ(ierr);
      for(j=istart;j<iend;j++){tempArrIdx[j-istart]=j;}
      ierr = MatSetValues(Kccstarinv,iend-istart,tempArrIdx,1,&i,tempArr,INSERT_VALUES);CHKERRQ(ierr);
      ierr = VecRestoreArray(tempVec3,&tempArr);CHKERRQ(ierr);
    }
    ierr = VecScatterDestroy(&vecscatNuc);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec2);CHKERRQ(ierr);
    ierr = VecDestroy(&tempVec3);CHKERRQ(ierr);
    ierr = MatDestroy(&tempMat2);CHKERRQ(ierr);
    ierr = PetscFree(tempArr);CHKERRQ(ierr);
    ierr = PetscFree(tempArrIdx);CHKERRQ(ierr);

    ierr = MatAssemblyBegin(Kccstarinv,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
    ierr = MatAssemblyEnd(Kccstarinv,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
    PetscPrintf(PETSC_COMM_WORLD," done.");

// ....................................................................

    PetscPrintf(PETSC_COMM_WORLD,"\nPreparing lambda equation: ");
    ierr = MatCreate(PETSC_COMM_WORLD,&tempMat1);CHKERRQ(ierr);
    ierr = MatSetSizes(tempMat1,PETSC_DECIDE,PETSC_DECIDE,Nlambda,Nuc);CHKERRQ(ierr);
    ierr = MatSetType(tempMat1,MATMPIDENSE);CHKERRQ(ierr);
    ierr = MatSetUp(tempMat1);CHKERRQ(ierr);
    ierr = MatMatMultMPI(FIrc,Kccstarinv,tempMat1);CHKERRQ(ierr);
    ierr = MatTranspose(FIrc,MAT_INITIAL_MATRIX,&tempMat2);CHKERRQ(ierr);
    ierr = MatCreate(PETSC_COMM_WORLD,&FKF);CHKERRQ(ierr);
    ierr = MatSetSizes(FKF,PETSC_DECIDE,PETSC_DECIDE,Nlambda,Nlambda);CHKERRQ(ierr);
    ierr = MatSetType(FKF,MATMPIDENSE);CHKERRQ(ierr);
    ierr = MatSetUp(FKF);CHKERRQ(ierr);
    ierr = MatMatMultMPI(tempMat1,tempMat2,FKF);CHKERRQ(ierr);  // computing matrix for lambda equation
    ierr = MatDestroy(&tempMat2);CHKERRQ(ierr);
    ierr = VecCreate(PETSC_COMM_WORLD,&rhs);CHKERRQ(ierr);
    ierr = VecSetSizes(rhs,PETSC_DECIDE,Nlambda);CHKERRQ(ierr);
    ierr = VecSetType(rhs,VECMPI);CHKERRQ(ierr);
    ierr = VecZeroEntries(rhs);CHKERRQ(ierr);
    ierr = MatMult(tempMat1,fcstar,rhs);CHKERRQ(ierr);
```

```
   ierr = VecAYPX(rhs,-1.0,dr);CHKERRQ(ierr);     // computing rhs vector for lambda equation
   ierr = VecDestroy(&dr);CHKERRQ(ierr);
   ierr = MatDestroy(&tempMat1);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD," done.");

// ..........................................................................
// creating empty global vectors lambda, uc

   ierr = VecCreate(PETSC_COMM_WORLD,&lambda);CHKERRQ(ierr);
   ierr = VecSetSizes(lambda,PETSC_DECIDE,Nlambda);CHKERRQ(ierr);
   ierr = VecSetType(lambda,VECMPI);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_WORLD,&uc);CHKERRQ(ierr);
   ierr = VecSetSizes(uc,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
   ierr = VecSetType(uc,VECMPI);CHKERRQ(ierr);

// ..........................................................................
// solving lambda equation by CG
// ..........................................................................

   PetscPrintf(PETSC_COMM_WORLD,"\nSolving lambda equation: ");
   PetscReal       g2old, g2new, alpha, beta, pAp;
   Vec             g, p, plocal, Ap;
   PetscInt        its;
   its = 0;
   ierr = VecDuplicate(rhs, &p );CHKERRQ(ierr);   // p  is created
   ierr = VecDuplicate(rhs, &Ap);CHKERRQ(ierr);   // Ap is created
   ierr = VecDuplicate(rhs, &g );CHKERRQ(ierr);   // g  is created
   ierr = VecSet(lambda, 0.0);CHKERRQ(ierr); // lambda=0
   ierr = VecSet(g, 0.0);CHKERRQ(ierr);      // g=A*lambda
   ierr = VecAYPX(g,-1.0,rhs);CHKERRQ(ierr); // g=rhs-g
   ierr = VecDestroy(&rhs);CHKERRQ(ierr);
   ierr = VecCopy(g, p);CHKERRQ(ierr);       // p=g
   ierr = VecDot(g,g,&g2old);CHKERRQ(ierr);  // g2old=(g,g)

   ierr = VecCreate(PETSC_COMM_SELF,&tempVec1);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Ni+Nr);CHKERRQ(ierr);
   ierr = VecSetType(tempVec1,VECSEQ);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec2);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec2,PETSC_DECIDE,Ni+Nr);CHKERRQ(ierr);
   ierr = VecSetType(tempVec2,VECSEQ);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec3);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec3,PETSC_DECIDE,Nlambda);CHKERRQ(ierr);
   ierr = VecSetType(tempVec3,VECSEQ);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_WORLD,&tempVec4);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec4,PETSC_DECIDE,Nlambda);CHKERRQ(ierr);
   ierr = VecSetType(tempVec4,VECMPI);CHKERRQ(ierr);

   while (g2old > 0.0000000001 && its <= 100)
   {
     ierr = PetscPrintf(PETSC_COMM_WORLD, "%d CG g2=%.10e \n", its, g2old);CHKERRQ(ierr);
     ierr = VecScatterCreateToAll(p,&vecscatToAll,&plocal);CHKERRQ(ierr);
     ierr = VecScatterBegin(vecscatToAll,p,plocal,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
     ierr = VecScatterEnd(vecscatToAll,p,plocal,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
     ierr = VecScatterDestroy(&vecscatToAll);CHKERRQ(ierr);
     ierr = MatMultTranspose(Br,plocal,tempVec1);CHKERRQ(ierr);
     ierr = VecDestroy(&plocal);CHKERRQ(ierr);
     ierr = MatMult(Krrinv,tempVec1,tempVec2);CHKERRQ(ierr);
     ierr = MatMult(Br,tempVec2,tempVec3);CHKERRQ(ierr);
     ierr = VecZeroEntries(tempVec4);CHKERRQ(ierr);
     ierr = VecScatterBegin(vecscatNlambda,tempVec3,tempVec4,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
     ierr = VecScatterEnd(vecscatNlambda,tempVec3,tempVec4,ADD_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
     ierr = MatMultAdd(FKF,p,tempVec4,Ap);CHKERRQ(ierr);
     ierr = VecDot(p,Ap,&pAp);CHKERRQ(ierr);           // pAp=p'*Ap
     alpha=g2old/pAp;                                  // alpha=g2old/pAp
     ierr = VecAXPY(lambda,alpha,p);CHKERRQ(ierr);     // lambda=lambda+alpha*p
     ierr = VecAXPY(g,-1.0*alpha,Ap);CHKERRQ(ierr);    // g=g-alpha*Ap
     ierr = VecDot(g,g,&g2new);CHKERRQ(ierr);          // g2new=(g,g)
     ierr = VecAYPX(p,g2new/g2old,g);CHKERRQ(ierr);    // p=g+(g2new/g2old)*p
     g2old=g2new;                                      // g2old=g2new
     its++;                                            // its=its+1
   }
   ierr = MatDestroy(&FKF);CHKERRQ(ierr);
   ierr = VecScatterDestroy(&vecscatNlambda);CHKERRQ(ierr);
   ierr = VecDestroy(&g);CHKERRQ(ierr);
   ierr = VecDestroy(&Ap);CHKERRQ(ierr);
   ierr = VecDestroy(&p);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec4);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec3);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec2);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD," done.");

// ..........................................................................

   PetscPrintf(PETSC_COMM_WORLD,"\nComputing uc from lambda: ");
   ierr = VecCreate(PETSC_COMM_WORLD,&tempVec1);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nuc);CHKERRQ(ierr);
   ierr = VecSetType(tempVec1,VECMPI);CHKERRQ(ierr);
   ierr = MatMultTranspose(FIrc,lambda,tempVec1);CHKERRQ(ierr);
   ierr = MatDestroy(&FIrc);CHKERRQ(ierr);
```

```
   ierr = VecAXPY(tempVec1,1.0,fcstar);CHKERRQ(ierr);
   ierr = VecDestroy(&fcstar);CHKERRQ(ierr);
   ierr = MatMult(Kccstarinv,tempVec1,uc);CHKERRQ(ierr);  // computing uc from lambda
   ierr = MatDestroy(&Kccstarinv);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD," done.");

// ....................................................................

   PetscPrintf(PETSC_COMM_WORLD,"\nComputing local vectors ur from uc and lambda: ");
   ierr = VecCreate(PETSC_COMM_SELF,&ur);CHKERRQ(ierr);
   ierr = VecSetSizes(ur,PETSC_DECIDE,Ni+Nr);CHKERRQ(ierr);
   ierr = VecSetType(ur,VECSEQ);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec1);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Ni+Nr);CHKERRQ(ierr);
   ierr = VecSetType(tempVec1,VECSEQ);CHKERRQ(ierr);
   ierr = MatMatMult(Krc,Bc,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&tempMat1);CHKERRQ(ierr);
   ierr = MatDestroy(&Krc);CHKERRQ(ierr);
   ierr = VecScatterCreateToAll(uc,&vecscatToAll,&tempVec3);CHKERRQ(ierr);
   ierr = VecScatterBegin(vecscatToAll,uc,tempVec3,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
   ierr = VecScatterEnd(vecscatToAll,uc,tempVec3,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
   ierr = VecDestroy(&uc);CHKERRQ(ierr);
   ierr = VecScatterDestroy(&vecscatToAll);CHKERRQ(ierr);
   ierr = MatMult(tempMat1,tempVec3,tempVec1);CHKERRQ(ierr);
   ierr = VecAYPX(tempVec1,-1.0,fr);CHKERRQ(ierr);
   ierr = VecDestroy(&fr);CHKERRQ(ierr);
   ierr = VecScatterCreateToAll(lambda,&vecscatToAll,&tempVec4);CHKERRQ(ierr);
   ierr = VecScatterBegin(vecscatToAll,lambda,tempVec4,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
   ierr = VecScatterEnd(vecscatToAll,lambda,tempVec4,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
   ierr = VecDestroy(&lambda);CHKERRQ(ierr);
   ierr = VecScatterDestroy(&vecscatToAll);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec2);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec2,PETSC_DECIDE,Ni+Nr);CHKERRQ(ierr);
   ierr = VecSetType(tempVec2,VECSEQ);CHKERRQ(ierr);
   ierr = MatMultTranspose(Br,tempVec4,tempVec2);CHKERRQ(ierr);
   ierr = MatDestroy(&Br);CHKERRQ(ierr);
   ierr = VecAXPY(tempVec1,-1.0,tempVec2);CHKERRQ(ierr);
   ierr = MatMult(Krrinv,tempVec1,ur);CHKERRQ(ierr);       // computing ur[] from uc and lambda
   ierr = MatDestroy(&Krrinv);CHKERRQ(ierr);
   ierr = MatDestroy(&tempMat1);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec4);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec2);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD," done.");

// ....................................................................

   PetscScalar *urArray;
   PetscScalar *ubcArray;
   PetscPrintf(PETSC_COMM_WORLD,"\nCreating local vectors u: ");
   ierr = VecCreate(PETSC_COMM_SELF,&u);CHKERRQ(ierr);
   ierr = VecSetSizes(u,PETSC_DECIDE,N);CHKERRQ(ierr);
   ierr = VecSetType(u,VECSEQ);CHKERRQ(ierr);
   ierr = VecGetArray(ur,&urArray);CHKERRQ(ierr);
   ierr = VecSetValues(u,Ni,ii,urArray,INSERT_VALUES);CHKERRQ(ierr);   // inserting values of interior nodes
   ierr = PetscFree(ii);CHKERRQ(ierr);
   ierr = VecSetValues(u,Nr,ir,urArray+Ni,INSERT_VALUES);CHKERRQ(ierr);  // inserting values of remainder nodes
   ierr = PetscFree(ir);CHKERRQ(ierr);
   ierr = VecRestoreArray(ur,&urArray);CHKERRQ(ierr);
   ierr = VecDestroy(&ur);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_SELF,&tempVec1);CHKERRQ(ierr);
   ierr = VecSetSizes(tempVec1,PETSC_DECIDE,Nc);CHKERRQ(ierr);
   ierr = VecSetType(tempVec1,VECSEQ);CHKERRQ(ierr);
   ierr = MatMult(Bc,tempVec3,tempVec1);CHKERRQ(ierr);
   ierr = MatDestroy(&Bc);CHKERRQ(ierr);
   ierr = VecGetArray(tempVec1,&ubcArray);CHKERRQ(ierr);
   ierr = VecSetValues(u,Nc,ic,ubcArray,INSERT_VALUES);CHKERRQ(ierr);   // inserting values of corner nodes
   ierr = PetscFree(ic);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec3);CHKERRQ(ierr);
   ierr = VecDestroy(&tempVec1);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD," done.");

// ....................................................................
// global vector of displacements

   ierr = VecCreate(PETSC_COMM_WORLD,&uGlobal);CHKERRQ(ierr);
   ierr = VecSetSizes(uGlobal,PETSC_DECIDE,NGHN*NGVN);CHKERRQ(ierr);
   ierr = VecSetType(uGlobal,VECMPI);CHKERRQ(ierr);
   ierr = VecZeroEntries(uGlobal);CHKERRQ(ierr);
   PetscScalar *uArray;
   PetscPrintf(PETSC_COMM_WORLD,"\nCreating global vector of displacements: ");
   PetscInt *temp;
   ierr = PetscMalloc(N*sizeof(PetscInt),&temp);CHKERRQ(ierr);l=0;
   for(j=csc[r][0];j<=csc[r][2];j++){
     for(k=csc[r][1];k<=csc[r][3];k++){
       temp[l]=j*NGHN+k;l++;
     }
   }
   ierr = VecGetArray(u,&uArray);CHKERRQ(ierr);                              // inserting displacements
   ierr = VecSetValues(uGlobal,N,temp,uArray,INSERT_VALUES);CHKERRQ(ierr);    // from local vector
```

```
   ierr = VecRestoreArray(u,&uArray);CHKERRQ(ierr);                          // to global vector
   ierr = VecDestroy(&u);CHKERRQ(ierr);
   ierr = PetscFree(temp);CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_WORLD," done.");
   ierr = VecAssemblyBegin(uGlobal);CHKERRQ(ierr);
   ierr = VecAssemblyEnd(uGlobal);CHKERRQ(ierr);

// ------------------------------ visualisation ---------------------------------

   ierr = VecScatterCreateToZero(uGlobal,&vecscatToZero,&uGlobalZero);CHKERRQ(ierr);
   ierr = VecScatterBegin(vecscatToZero,uGlobal,uGlobalZero,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
   ierr = VecScatterEnd(vecscatToZero,uGlobal,uGlobalZero,INSERT_VALUES,SCATTER_FORWARD);CHKERRQ(ierr);
   ierr = VecScatterDestroy(&vecscatToZero);CHKERRQ(ierr);    ierr = VecDestroy(&uGlobal);CHKERRQ(ierr);
   if(r==0){
      PetscDrawCreate(PETSC_COMM_SELF,PETSC_NULL,"Coloured solution",0,0,600,600,&draw);
      PetscDrawSetType(draw,PETSC_DRAW_X);
      VecGetArray(uGlobalZero,&uArray);
      PetscDrawTensorContour(draw,NGHN,NGVN,PETSC_NULL,PETSC_NULL,uArray);
      PetscDrawSetPause(draw,-1);
      PetscDrawPause(draw);
   }
   ierr = VecDestroy(&uGlobalZero);CHKERRQ(ierr);

   PetscPrintf(PETSC_COMM_WORLD,"\nALL DONE.\n\n");

   PetscFinalize();
   return 0;
}
// ------------------------------ end of main function --------------------------


// ------------------------------------------------------------------------------
// ------------------------------ functions -------------------------------------
// ------------------------------------------------------------------------------
int MatMatMultMPI(Mat A,Mat B,Mat C)  // A*B=C
{    PetscInt i,j,ncolB,nrowB,nrowC,istart,iend;
   Vec colB,colC;
   PetscScalar *tempArr;
   PetscInt *tempArrIdx;

   ierr = MatGetSize(B,&nrowB,&ncolB);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_WORLD,&colB);CHKERRQ(ierr);
   ierr = VecSetSizes(colB,PETSC_DECIDE,nrowB);CHKERRQ(ierr);
   ierr = VecSetType(colB,VECMPI);CHKERRQ(ierr);
   ierr = MatGetSize(C,&nrowC,PETSC_NULL);CHKERRQ(ierr);
   ierr = VecCreate(PETSC_COMM_WORLD,&colC);CHKERRQ(ierr);
   ierr = VecSetSizes(colC,PETSC_DECIDE,nrowC);CHKERRQ(ierr);
   ierr = VecSetType(colC,VECMPI);CHKERRQ(ierr);
   for(i=0;i<ncolB;i++){
     ierr = MatGetColumnVector(B,colB,i);CHKERRQ(ierr);
     ierr = MatMult(A,colB,colC);CHKERRQ(ierr);
     ierr = VecGetOwnershipRange(colC,&istart,&iend);CHKERRQ(ierr);
     ierr = PetscMalloc((iend-istart)*sizeof(PetscScalar),&tempArr);CHKERRQ(ierr);
     ierr = PetscMalloc((iend-istart)*sizeof(PetscInt),&tempArrIdx);CHKERRQ(ierr);
     ierr = VecGetArray(colC,&tempArr);CHKERRQ(ierr);
     for(j=istart;j<iend;j++){tempArrIdx[j-istart]=j;}
     ierr = MatSetValues(C,iend-istart,tempArrIdx,1,&i,tempArr,INSERT_VALUES);CHKERRQ(ierr);
     ierr = VecRestoreArray(colC,&tempArr);CHKERRQ(ierr);
     ierr = PetscFree(tempArr);CHKERRQ(ierr);
     ierr = PetscFree(tempArrIdx);CHKERRQ(ierr);
   }    ierr = MatAssemblyBegin(C,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatAssemblyEnd(C,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
   ierr = VecDestroy(&colB);CHKERRQ(ierr);
   ierr = VecDestroy(&colC);CHKERRQ(ierr);
return 0;
}
// ................................................................................

void FEM(Mat Ks, Vec fs,PetscBool *lsa, PetscInt vSize, PetscInt hSize)
{
   PetscInt i,j;
   for(i=0;i<(vSize-1);i++){
     for(j=0;j<(hSize-1);j++){
       if((*(lsa+i*hSize+j))&&(*(lsa+i*hSize+j+1))&&(*(lsa+(i+1)*hSize+j))&&(*(lsa+(i+1)*hSize+j+1))){
         Add2Triangles(Ks,fs,i*hSize+j,i*hSize+j+1,(i+1)*hSize+j,(i+1)*hSize+j+1);
       }
     }
   }
}
// ................................................................................

int Add2Triangles(Mat Ks, Vec fs, PetscInt a, PetscInt b, PetscInt c, PetscInt d)
{
   ierr = MatSetValue(Ks,a,a,1.0,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,a,b,-0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,a,c,-0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,b,a,-0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,b,b,0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,c,a,-0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,c,c,0.5,ADD_VALUES);CHKERRQ(ierr);
```

```
   ierr = VecSetValue(fs,a,1.0/6.0,ADD_VALUES);CHKERRQ(ierr);
   ierr = VecSetValue(fs,b,1.0/6.0,ADD_VALUES);CHKERRQ(ierr);
   ierr = VecSetValue(fs,c,1.0/6.0,ADD_VALUES);CHKERRQ(ierr);

   ierr = MatSetValue(Ks,d,d,1.0,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,d,b,-0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,d,c,-0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,b,d,-0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,b,b,0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,c,d,-0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = MatSetValue(Ks,c,c,0.5,ADD_VALUES);CHKERRQ(ierr);
   ierr = VecSetValue(fs,d,1.0/6.0,ADD_VALUES);CHKERRQ(ierr);
   ierr = VecSetValue(fs,b,1.0/6.0,ADD_VALUES);CHKERRQ(ierr);
   ierr = VecSetValue(fs,c,1.0/6.0,ADD_VALUES);CHKERRQ(ierr);
}
// .....................................................................

int DirichletBoundaryCondition(Mat Ks, Vec fs, PetscInt n, PetscReal value)
{
   PetscInt i,nRowK,nColK;
   ierr = MatGetSize(Ks,&nRowK,&nColK);CHKERRQ(ierr);
   ierr = MatAssemblyBegin(Ks,MAT_FLUSH_ASSEMBLY);CHKERRQ(ierr);
   ierr = MatAssemblyEnd(Ks,MAT_FLUSH_ASSEMBLY);CHKERRQ(ierr);
   for(i=0;i<nRowK;i++) MatSetValue(Ks,n,i,0.0,INSERT_VALUES);
   for(i=0;i<nColK;i++) MatSetValue(Ks,i,n,0.0,INSERT_VALUES);
   ierr = MatSetValue(Ks,n,n,1.0,INSERT_VALUES);CHKERRQ(ierr);
   ierr = VecSetValue(fs,n,value,INSERT_VALUES);CHKERRQ(ierr);
}
// .....................................................................

PetscInt BelongsToSubdomain(PetscInt row,PetscInt col)
{
   PetscInt i;
   for(i=0;i<Ns;i++){
     if((row>=csc[i][0])&&(row<=csc[i][2])&&(col>=csc[i][1])&&(col<=csc[i][3])){
       return i;
     }
   }
   return -1;
}
```

Bibliography

[BBB+12] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2012.

[Bou10] Jiri Bouchala. Variacni metody, 2010.

[CF01] Patrick LeTallec Kendall Pierson Daniel Rixen Charbel Farhat, Michel Lesoinne. Feti-dp: a dual primal unified feti method -part i: A faster alternative to the two-level feti method. *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING*, 50: 1523–1544, 2001.

[Dos05] Stefanica Dostal, Horak. A scalable feti-dp algorithm for a coercive variational inequalities, 2005.

[Dos06] Stefanica Dostal, Horak. An overview of scalable feti-dp algorithms for variational inequalities, 2006.

[Hor07] David Horak. *FETI based domain decomposition methods for variational inequalities*. PhD thesis, VSB - Technical University Ostrava, 2007.

[Str08a] Gilbert Strang. Lecture 17: Finite elements in 1d (part 1), 2008.

[Str08b] Gilbert Strang. Lecture 18: Finite elements in 1d (part 2), 2008.

[Str08c] Gilbert Strang. Lecture 23: Laplace's equation, 2008.

[Str08d] Gilbert Strang. Lecture 24: Laplace's equation (part 2), 2008.

[Str08e] Gilbert Strang. Lecture 26: Fast poisson solver (part 2); finite elements in 2d, 2008.

[Str08f] Gilbert Strang. Lecture 27: Finite elements in 2d (part 2), 2008.

Nomenclature

λ	Lagrange multipliers vector
Ω^s	subdomain
Φ_j	trial function
B	Boolean matrix
C	constant of integration
F	dual operator
f	load vector
K	stiffness matrix
u	displacement
u_c	global vector of corners
v	test function